Interfaces Cérebro-Computador: Processamento de Sinal e Reconhecimento de Padrões

João Ricardo Alexandre Isidoro Cabrita

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Júri
Presidente: Professor Marcelino Bicho dos Santos
Orientador: Professor Pedro Filipe Zeferino Tomás
Vogais: Professor João Miguel Raposo Sanches

December 6, 2010
Acknowledgments

I would like to thank my supervisor, Dr. Pedro Tomás, for his helpful insight and guidance during the development of the work presented in this thesis. I would also like to acknowledge Prof. Leonel Sousa for changing my views on my academic life and taking me in as a member of the SIPS group.

A heartfelt thank you goes out to my family, who have supported me throughout my entire life and without whom I would not be where I am today.

I would also like to thank my girlfriend Tânia for cheering me on for nearly all of my academic course.

Thanks to all of my friends, particularly Filipa, who I’ve known since the first year of college and has helped me through tough times.

Finally, I would like to express my appreciation for Rosa and José from Deloitte for giving me the freedom to balance work on my thesis and my job.
Abstract

Recent technological advances have allowed Brain-Computer Interfaces (BCIs), systems capable of capturing and interpreting human brain activity, to surface as a popular research trend in recent years. Particularly when paired with the Electroencephalogram (EEG), BCI systems show great promise in the field of rehabilitation engineering. However, the performance of such systems is still limited and largely subject dependent and further improvements must be made to provide increased performance. Moreover, the lack of a structured environment for researching and developing in this area makes benchmarking and comparing different approaches a highly subjective matter. To address these issues, this thesis starts by introducing basic concepts in EEG-based BCI, namely neurophysiology and the popular Common Spatial Patterns (CSP) method and two derived approaches, Common Spatio-Spectral Patterns (CSSP) and Common Sparse Spectral Spatial Patterns (CSSSP). Then, two modifications which improve these algorithms are made, the first dealing with the selection of filters computed using the mentioned methods and the second with increasing Signal to Noise Ratio (SNR) through tuning of the pre-processing stages. Three new approaches are also introduced, Combined Frequency Band Spatial Patterns (CFBSP), Spatial Filter Residual Analysis (SFRA) and Spatial Patterns for Maximal Separability (SPMS), which use the same basic concepts as the algorithms above but present novel ways of exploring the data. Finally, a MATLAB framework for developing and testing BCI algorithms is presented, focused on quick prototyping by reusing code and equally testing different algorithms. Testing the above proposed modifications and methods showed that these effectively improve on the performance of plain CSP, despite still having a large margin for being improved.

Keywords

Brain-Computer Interface, Common Spatial Patterns, MATLAB Framework, Electroencephalogram, Signal Processing
Resumo

Os avanços tecnológicos dos últimos anos levaram ao aparecimento das Interfaces Cérebro-Computador (BCIs), sistemas capazes de adquirir e interpretar a actividade do cérebro humano, como uma área de investigação atractiva. Particularmente quando o sinal de entrada é o Electroencefalograma (EEG), este tipo de sistemas demonstra grande potencial em aplicações como a prostética de reabilitação. No entanto, a capacidade destes sistemas de interpretar as intenções do utilizador são ainda muito limitadas. Além disso, a falta de um ambiente estruturado para desenvolver e testar algoritmos deste tipo torna difícil comparar abordagens diferentes. Para mitigar estes problemas, esta tese começa por introduzir conceitos básicos em BCIs baseados em EEG, apresentando o método Common Spatial Patterns (CSP) e dois métodos derivados deste, Common Spatio-Spectral Patterns (CSSP) e Common Sparse Spectral Spatial Patterns (CSSSP). São então propostas duas modificações a estes algoritmos com vista a melhorar a seleção dos filtros calculados por estes métodos e aumentar a Signal to Noise Ratio (SNR) do sinal de entrada através da afinação dos componentes de pré-processamento. São também apresentados três novos métodos, Combined Frequency Band Spatial Patterns (CFBSP), Spatial Filter Residual Analysis (SFRA) e Spatial Patterns for Maximal Separability (SPMS), que utilizam como base os algoritmos mencionados acima mas que exploram os dados de forma inovadora. Finalmente, é apresentada uma estrutura de desenvolvimento e teste de algoritmos BCI em ambiente MATLAB que privilegia o desenvolvimento rápido através da reutilização de código e o teste de diferentes algoritmos em igualdade de circunstâncias. Após o teste das modificações e métodos propostos acima, confirmou-se que estes apresentam performance superior à do CSP.

Palavras Chave

Interfaces Cérebro-Computador, Electroencefalograma, Processamento de Sinal
Contents

1 Introduction ............................................. 1
   1.1 Objectives ........................................... 3
   1.2 Main contributions ................................... 3
   1.3 Dissertation outline .................................. 4

2 Basics of Brain-Computer Interfacing .......................... 5
   2.1 Neurophysiology ...................................... 6
   2.2 Recording of Brain Signals ............................. 7
      2.2.1 Electroencephalography ............................ 8
   2.3 Structure of Brain-Computer Interfaces (BCIs) ............... 9
   2.4 State of the art ....................................... 11

3 Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs) 13
   3.1 Algorithms for Brain-Computer Interfaces (BCIs) ............... 14
      3.1.1 Common Spatial Patterns (CSP) ...................... 15
      3.1.2 Common Spatio-Spectral Patterns (CSSP) ............... 20
      3.1.3 Common Sparse Spectral Spatial Patterns (CSSSP) .......... 23
   3.2 Selection of Spatial Filters ................................ 25
   3.3 Frequency Band Selection ................................ 26
   3.4 Combined Frequency Band Spatial Patterns (CFBSP) ............... 28
   3.5 Spatial Patterns for Maximal Separability (SPMS) .............. 30
   3.6 Spatial Filter Residual Analysis (SFRA) ..................... 32

4 BCILAB — Brain-Computer Interface (BCI) MATLAB Library ......... 35
   4.1 Core functions ........................................ 36
      4.1.1 Simulator skeleton .................................. 36
      4.1.2 EEG_COV function .................................. 37
      4.1.3 EEG_CSP function .................................. 38
      4.1.4 EEG_FILTER function ............................... 39
      4.1.5 PICKFILTERS function .............................. 40
4.2 Algorithm Implementations ........................................ 40
  4.2.1 Common Spatial Patterns (CSP) ................................ 41
    A  parse_args ................................................. 41
    B  read_data ................................................. 42
    C  global_prep .............................................. 43
    D  local_prep .............................................. 43
    E  get_params ............................................... 43
    F  get_features ............................................. 45
    G  run_classifier ........................................... 45
    H  get_run_results .......................................... 47
    I  package_results .......................................... 47
  4.2.2 Common Spatio-Spectral Patterns (CSSP) .................... 48
    A  parse_args ................................................. 48
    B  read_data ................................................. 48
    C  package_results .......................................... 48
  4.2.3 Common Sparse Spectral Spatial Patterns (CSSSP) ......... 49
    A  parse_args ................................................. 49
    B  local_prep .............................................. 49
    C  get_params ............................................... 49
    D  get_run_results .......................................... 49
    E  package_results .......................................... 50
  4.2.4 Combined Frequency Band Spatial Patterns (CFBSP) ....... 50
    A  parse_args ................................................. 50
    B  local_prep .............................................. 50
  4.2.5 Spatial Patterns for Maximal Separability (SPMS) ......... 50
    A  get_params ............................................... 50
  4.2.6 get_features ................................................ 50
  4.2.7 Spatial Filter Residual Analysis (SFRA) .................... 52
    A  get_features ............................................. 52

5 Experimental results .................................................. 55
  5.1 Testing framework .............................................. 56
  5.2 Algorithm results .............................................. 57
    5.2.1 Common Spatial Patterns (CSP) ........................ 57
    5.2.2 Common Spatio-Spectral Patterns (CSSP) ............... 62
    5.2.3 Common Sparse Spectral Spatial Patterns (CSSSP) ...... 65
    5.2.4 Combined Frequency Band Spatial Patterns (CFBSP) .... 69
    5.2.5 Spatial Filter Residual Analysis (SFRA) ............... 73
5.2.6 Spatial Patterns for Maximal Separability (SPMS) .......................... 77
5.3 Effects of Frequency Band Selection ............................................. 78
5.4 Comparison of classifiers ......................................................... 80
5.5 Comparison of BCI algorithms ................................................... 80

6 Conclusions ................................................................................. 89
   6.1 Result Review ................................................................. 90
   6.2 Future work ................................................................. 91
List of Figures

2.1 Structure of the human brain. ............................................. 6
2.2 A subject wearing an Electroencephalogram (EEG) cap. ............ 8
2.3 Structure of a BCI. .......................................................... 9
2.4 Examples of time structures for trials without feedback and with feedback. ................................................. 10
3.1 Diagram for the training and testing phases of CSP. ................ 16
3.2 Example EEG signals for right hand and foot Motor Imagerys (MI). The legend is the value of the signals variance. ......................................................... 17
3.3 Diagram for the training and testing phases of CSSP. ............... 21
3.4 Diagram for the training and testing phases of CSSSP. ............. 24
3.5 Trees built for band selection. ............................................. 27
3.6 An example of running band selection. .................................. 28
3.7 Diagram for the training and testing phases of CFBSP. .............. 29
3.8 Diagram for the training and testing phases of SPMS. ............... 30
3.9 Diagram for the training and testing phases of SFRA. ............... 33
4.1 Simplified code of the BCI skeleton function. ......................... 36
4.2 Simplified code for the eeg_cov function. ............................... 38
4.3 Simplified code for the eeg_csp function. ............................... 38
4.4 Simplified code for the eeg_filter function. ........................... 39
4.5 Simplified code for the pickfilters function. .......................... 40
4.6 Simplified code for the parse_args function. .......................... 42
4.7 Simplified code for the read_data function. ............................ 42
4.8 Simplified code for the global_prep function. .......................... 43
4.9 Simplified code for the local_prep function. ............................ 44
4.10 Simplified code for the get_params function. .......................... 44
4.11 Simplified code for the get_features function. ........................ 45
4.12 Simplified code for the run_classifier function. ........................ 46
4.13 Simplified code for the read_data function of the CSSP implementation. ............................................. 48
4.14 Simplified code for the get_params function in the CSSSP implementation. ............................................. 49
List of Figures

4.15 Simplified code for the local_prep function implementation for the CFBSP algorithm. 51
4.16 Simplified code for the SPMS implementation of the get_params function. 51
4.17 Simplified code for the SFRA implementation of the get_features function. 52

5.1 Spatial Filters for each MI obtained through the CSP method. 57
5.2 BCI classification results for the CSP algorithm, using the traditional spatial filter selection method and no frequency tuning. 58
5.3 BCI classification results for the CSP algorithm, using the mixed spatial filter selection method and no frequency tuning. 59
5.4 BCI classification results for the CSP algorithm, using the sorted spatial filter selection method and no frequency tuning. 59
5.5 Comparison of filter selection methods with no frequency band tuning applied, for the CSP algorithm. 61
5.6 Variation in the accuracy of the CSSP algorithm when different time delays are used. 63
5.7 Comparison of filter selection methods with no frequency band tuning applied, for the CSSP method. 64
5.8 Spatial Filters for each MI and Spectral Filter obtained through the CSSSP method. 66
5.9 BCI classification results for the CSSSP algorithm, using the traditional spatial filter selection method. 67
5.10 BCI classification results for the CSSSP algorithm, using the mixed spatial filter selection method. 67
5.11 BCI classification results for the CSSSP algorithm, using the sorted spatial filter selection method. 67
5.12 Comparison of filter selection methods with no frequency band tuning applied, for the CSSSP algorithm. 68
5.13 Spatial filters obtained using CFBSP for each of the frequency bands and MIs. 69
5.14 BCI classification results for the CFBSP algorithm, using the traditional spatial filter selection method and no frequency tuning. 70
5.15 BCI classification results for the CFBSP algorithm, using the mixed spatial filter selection method and no frequency tuning. 70
5.16 BCI classification results for the CFBSP algorithm, using the sorted spatial filter selection method and no frequency tuning. 71
5.17 Comparison of filter selection methods for the CFBSP method. 72
5.18 Spatial filters obtained using the SFRA algorithm, for each of the MIs. The bars on the right-hand side of each spatial filter map the colours to the numerical weights given to channels. 73
5.19 Predictors obtained using the SFRA algorithm, for each of the MIs. 74
5.20 BCI classification results for the SFRA algorithm, using the traditional spatial filter selection method and no frequency tuning. ........................................ 75
5.21 BCI classification results for the SFRA algorithm, using the mixed spatial filter selection method and no frequency tuning. ................................. 75
5.22 BCI classification results for the SFRA algorithm, using the sorted spatial filter selection method and no frequency tuning. ............................. 75
5.23 Comparison of filter selection methods with no frequency band tuning applied, for the SFRA method. ................................................................. 76
5.24 Spatial Filters for each MI obtained through the SPMS method. The bars on the right-hand side of each filter map the colours to the numerical weights given to channels. ................................................................. 77
5.25 BCI classification results for the SPMS algorithm. ................................ 77
5.26 Comparison of CSP and Frequency Tuned (FT)CSP. .......................... 78
5.27 Comparison of CSSP and FTCSSP. .................................................... 79
5.28 Comparison of SFRA and FTSFRA. ................................................... 79
5.29 Comparison of classifiers for the CSP algorithm. ................................ 81
5.30 Comparison of CSP and CSSP. .......................................................... 82
5.31 Comparison of CSP and CSSSP. ....................................................... 82
5.32 Comparison of CSP and CFBSF. ....................................................... 83
5.33 Comparison of CSP and SFRA. .......................................................... 83
5.34 Comparison of CSP and SPMS. .......................................................... 84
5.35 Comparison of the maximum obtained accuracy for several BCI algorithms. ................................. 85
5.36 Comparison of the obtained accuracy for several BCI algorithms when the number of spatial filters is restricted to 2. ................................. 85
List of Tables

4.1 Functions called by the skeleton function that can be overridden, together with their intended use. ................................................................. 37

5.1 Number of delayed samples in the CSSP algorithm for each subject. .............. 80
5.2 Simulation parameters for each algorithm of Figure 5.35. ............................ 86
List of acronyms

BCI  Brain-Computer Interface
SNR  Signal to Noise Ratio
MI   Motor Imagery
EEG  Electroencephalogram
ECoG Electrocorticography
fMRI Functional Magnetic Resonance Imaging
MEG  Magnetoencephalogram
ERS  Event-Related Synchronization
ERD  Event-Related Desynchronization
CSP  Common Spatial Patterns
CSSP Common Spatio-Spectral Patterns
CSSSP Common Sparse Spectral Spatial Patterns
CFBSP Combined Frequency Band Spatial Patterns
SPMS Spatial Patterns for Maximal Separability
SFRA Spatial Filter Residual Analysis
LDA  Linear Discriminant Analysis
NBC  Naïve Bayes Classifier
SVM  Support Vector Machine
AR   Auto-Regressive
FIR  Finite Impulse Response
IIR  Infinite Impulse Response
FT   Frequency Tuned
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
</tr>
</thead>
</table>

xviii
1 Introduction

Contents

1.1 Objectives ......................................................... 3
1.2 Main contributions .............................................. 3
1.3 Dissertation outline ............................................. 4
1. Introduction

Instrumentation advances in the recording and processing of brain activity have lead to the recent appearance of Brain-Computer Interfaces (BCIs) as a popular research trend. A BCI is a system which allows control of electronic devices by interpretation of brain signals. This opens up many possibilities such as developing artificial limbs that can be used to replace normal limbs lost in an accident, spelling into a computer by using thought alone, or remotely operating machines.

In general, the setup and operation of a BCI is structured as other common supervised machine learning applications, that is, it relies on a training step in which data labelled with the desired system outcome is supplied to the system for constructing a model. This model is then used for classifying the data fed into the system when it is in operation. Since organizing the training and operation in this way doesn’t make any assumptions about the employed model, there is a large amount of flexibility allowed for choosing models. There is also a great variety of methods used to extract data, ranging from invasive methods like brain implants and Electroencephalography (EEG), which require surgery, to non-invasive ones like fMRI and Magnetoencephalograhm (MEG) which require much less set-up but acquire a noisier signal with less spatial and temporal resolution. Although all these methods can be used in practice to develop BCIs, EEG is by far the most popular, particularly when used with the CSP method for feature extraction. This method draws on the specifics of brain activity related to motor tasks and has become a prime candidate for applications like commanding a prosthetic arm or navigating through computer interfaces.

The MSc thesis presented here results from the desire of the SIPS research group (a part of INESC-ID) to enter this new research area and develop a functional BCI and was developed over the course of a year, from September 2009 to September 2010. This BCI can be broadly separated into hardware and software components, with the hardware portion handling the acquisition of signal from the subject, preparing it to be processed by the software component and later driving the decision made by the software component to the device which the BCI is to control. The software component, which is the focus of this thesis, is in this way left with the responsibility of interpreting the incoming signal (by using the CSP method mentioned above) and producing a guess which should coincide with the user’s intentions. Among the great number of challenges involved in the development of this component (of which only a fraction could be reasonably addressed in this thesis) are:

1. the gathering of a baseline of information on BCIs and selection of reference methods;
2. the creation of an environment which allows for quick and easy prototyping of method implementations;
3. the benchmarking of those implementations.

The first of these challenges translates into knowing how to extract signals from the brain, how these signals are structured and what their interesting characteristics are and how these are
1.1 Objectives

Currently being used. This means that a variety of backgrounds is involved in BCI development, ranging from neurophysiological to signal processing and machine learning.

The second challenge addresses the effort required to develop for custom-built platforms. A way to solve this problem is to perform this development effort on a standardized platform first and then port it to suit the final device. This implementation workflow thus emphasizes design over implementation since it is easier to materialize ideas as opposed to becoming lost on the details of the platform.

Finally, the challenge of benchmarking is closely related to the previous one in the sense that the benefits of quick prototyping are greatly increased when it’s possible to benchmark various prototypes. This happens because benchmarking matches each prototype with a measure of performance and provides a foundation for deciding which prototypes are worth the additional cost of porting to the custom device.

1.1 Objectives

The objective of this dissertation is to study and document the basics of CSP-based BCI theory, implement and compare several existing BCI algorithms and propose changes in these algorithms which lead to improved performance.

1.2 Main contributions

The work associated with this thesis contributes by providing a baseline of information on CSP-based BCIs, namely by presenting the generic BCI architecture and describing the phenomena in the brain which are relevant for BCIs. It also presents and analyses three important CSP-based BCI algorithms proposed in literature, which use either spatial information only or both spatial and temporal information. These methods rely on the computation of several filters but use only a subset of those, selected using a criterion which is not always optimal. Thus, two new criteria for filter selection are proposed and tested which can improve performance over the traditional one. A frequency band selection procedure was also developed to tune the pre-processing filter and improve the Signal to Noise Ratio (SNR) of the input signal, allowing increased accuracy.

Additionally, three new methods for use in the BCI context were created. The first of these methods slightly modifies the frequency band selection procedure mentioned above to filter the input signal into several frequency bands which are processed together, while the second employs filtering of signals derived from Auto-Regressive (AR) models and the last attempts to optimize the separation of the distribution of features for classification.

Finally, a collection of MATLAB functions was developed to allow the quick prototyping and testing of BCI algorithms. This library provides a basic framework for modifying existing algorithms.
1. Introduction

or developing entirely new ones with reduced effort, was used to simulate and compare the above
mentioned CSP-based algorithms and can provide the basis for a working real-world BCI.

1.3 Dissertation outline

The remainder of this dissertation is organized as follows: Chapter 2 contains basic notions
and an overview of the state of the art in the field of BCIs. In Chapter 3, the theoretical work done
in the scope of this thesis is presented. This consists of a thorough analysis of three existing
BCI algorithms, presentation of three new algorithms and also the introduction of two algorithm
modifications. Chapter 4 deals with the implementation of the BCI simulator, describing in detail
it’s architecture as well as that of the MATLAB library developed in the scope of this thesis. Finally,
Chapter 5 presents the experimental procedures followed and their results which are then used
to draw the conclusions in Chapter 6.
2 Basics of Brain-Computer Interfacing

Contents

2.1 Neurophysiology .......................................................... 6
2.2 Recording of Brain Signals .......................................... 7
2.3 Structure of Brain-Computer Interfaces (BCIs) ................. 9
2.4 State of the art ............................................................. 11
2. Basics of Brain-Computer Interfacing

In this chapter, some basic notions of how the brain works, how signals are extracted from it and how a Brain-Computer Interface (BCI) is structured are presented, along with the state of the art in this area of research. Although not essential to understanding the methods in the next chapter, this chapter will clarify the type of information extracted from the brain which is relevant in the context of Common Spatial Patterns (CSP)-based BCIs and how the BCIs use this information to classify brain activity.

2.1 Neurophysiology

The brain is the center of the human nervous system and a very complex organ. It is the entity responsible for many of the autonomic processes of the body such as the heart beat and breathing rates, as well as other higher functions such as learning, memory and motor coordination.

A diagram of the brain’s structure is presented in Figure 2.1.

![Figure 2.1: Structure of the human brain. Source: [1]](image)

In this diagram, one can observe that the brain can be partitioned according to function. Although it is not shown in the diagram, the brain is also split into two almost functionally symmetrical hemispheres, i.e., one area of the brain’s left hemisphere performs a similar function to the respective area in the right hemisphere. Of all the presented areas, the most important in terms of CSP-based BCI operation is the motor cortex, as it is over this area that relevant brain activity is located [2, 3].

In particular, activity in the motor cortex is related to motor tasks (such as moving arms and...
2.2 Recording of Brain Signals

legs), which are strongly connected to the CSP-based BCI experimental paradigm through the concept of MI [2]. The term MI refers to the imagination of performing a motor task without that task actually being performed. It has been shown that brain waves from actually performed tasks differ only slightly from those generated by imagined tasks and, as such, one can harness pre-existing knowledge about brain activity from motor tasks to the development of BCIs [2].

Another interesting characteristic of the human brain is its lateralization. As stated before, there is a functional symmetry in the human brain. However, each side of the brain tends to interact mainly with one of the sides of the body and, oddly enough, the left brain hemisphere controls the right side of the body while the right hemisphere controls the left side. It is this particular way of interaction between brain and body that we call lateralization, and this phenomenon provides some motivation to the CSP approach, as it can provide a spatial differentiation between mental states. Recalling the concept of MI, one can consider two imagined tasks: movement of the left arm and movement of the right arm. We would then expect that, while imagining movement for the left arm, brain activity over the left side of the brain is reduced while it increases for the right side and vice-versa for the MI of the right arm [2].

2.2 Recording of Brain Signals

The study of brain activity requires us to first develop ways in which this activity can be measured. This has led to the development of a great deal of equipment able to measure quantities which relate to brain activity. Examples of quantities which are known to be associated with brain functions are electrical currents over the cerebral cortex and the scalp as well as magnetic fields and blood flow in the brain, and all of these can be used to implement BCIs [4, 5].

In addition to discriminating BCIs using the quantity which is being measured, one can also split them into categories according to how invasive the method for recording signals is. We can thus define three categories of methods [5]:

- **Invasive**, in which at least the acquisition device has to be implanted in the grey matter of the brain, requiring surgery;
- **Semi-invasive**, which, although requiring surgery to implant the BCI inside the skull, position the acquisition device between the skull and the grey matter [6];
- **Non-invasive**, which collect the signals of interest with no need for surgery.

This categorization implies a trade-off in terms of health risks/practicality and quality, with invasive BCIs posing the greater risk in terms of health (infection of the brain) but also possessing greater signal quality due to proximity with the neurons generating the signals, and non-invasive ones causing nearly no risk to the user but suffering from degraded signal quality caused by the skull’s attenuation of the signals.
2. Basics of Brain-Computer Interfacing

This very reduced risk for the user of non-invasive BCIs seems to be the main explanation to why most developed BCIs are non-invasive. In particular, one non-invasive method of signal acquisition, electroencephalography, is arguably the most popular, as it combines the typical non-invasive BCI characteristics with high portability and accessible cost. The signal which is acquired through electroencephalography is called the Electroencephalogram (EEG). For the above reasons and because it is the basis for CSP methods, this thesis focuses on the use of the EEG.

2.2.1 Electroencephalography

The EEG relies on reading electrical currents originating from neuron activity which propagate to the scalp, where they are picked up by typically an array of electrodes in a cap (see Figure 2.2). The fact that the electrodes are positioned over the scalp results in poor spatial resolution (inaccuracy in spatially locating events) in EEG measurements, because, as stated before, there is some distance between the electrode and the current generating neurons and thus the currents tend to spread out, attenuate and cancel each others out. This is balanced by the fact that EEG has very good temporal resolution (on the order of milliseconds) and is very well studied [7].

Figure 2.2: A subject wearing an EEG cap.

The EEG is very important in the context of CSP-based BCIs because of the very noticeable phenomena observed during MI-tasks (see Section 2.1). These phenomena are called Event-Related Synchronization (ERS) and Event-Related Desynchronization (ERD), and relate to an increase and decrease, respectively, of the amplitude in the oscillations of the brain signal caused by the activity of a group of neurons [8]. Although ERS and ERD aren’t exclusive to MI-tasks, the lateralization of the brain provides for the interesting observation that, while performing a specific MI task, ERD tends to occur on one side of brain at the same time ERS occurs on the opposite side of the brain. If another MI task were performed with the opposite part of the body with which the first MI task was performed, the locations of ERD and ERS would have been switched.

Rather than being defined for the full frequency spectrum, one usually specifies both spatial and spectral locations of ERD and ERS, with the spectral location being one of several commonly designated frequency bands. A usual frequency domain partition is the following [9]:

8
- Delta, which comprises frequencies up to 4 Hz;
- Theta, the band over the frequency range [5, 7] Hz;
- Alpha, covering the frequencies between 8 and 12 Hz;
- Beta, for frequencies in [12, 30] Hz;
- Gamma, relating to the frequencies which go from 30 Hz up.

An additional denomination for Alpha band activity exists for singling out the portion of brain activity that occurs specifically over the motor cortex and in the [8, 12] Hz band, which is the Mu rhythm. This type of oscillation, along with the Beta frequency range, is very important for CSP-based BCI.

### 2.3 Structure of Brain-Computer Interfaces (BCIs)

BCIs are structured much like many supervised machine learning problems. This structure is presented in Figure 2.3

![Figure 2.3: Structure of a BCI.](image)

An initial set of data (training data) is supplied to the BCI, which might or might not employ some pre-processing steps, such as frequency filtering or artifact removal techniques (movement of the eyes or muscles of the face commonly induce artefactual activity in the EEG). The frequency filtering is usually included as an artifact removal technique since typical EEG artefacts appear in low frequency ranges. This is followed by feature extraction, an operation or a series of operations which transform the data into a feature or feature array. The goal of this step is to accentuate the characteristics of the data which differ among the classes of the machine learning problem at hand (in the BCI context, the term classes refers to the set of MI tasks which the BCI is expected to distinguish). The features obtained from the training data are then used to train a classifier (i.e., create a model from these features such that, if supplied another set of features, it outputs
their class labels), thus ending the training phase. In the BCI context, several approaches exist for extracting features [2] such as Band Power Estimation, Adaptive Autoregressive Models, Common Spatial Patterns (CSP) and Hidden Markov Models. The most popular approach is the CSP one [2, 3, 7, 10–13] and as such research for this thesis is concentrated on this method.

After these procedures, the BCI is ready to be used (tested). The test phase is composed of a pre-processing stage similar to the one used in training followed by feature extraction. After this, the features are fed to the classifier that was previously trained in the training phase, producing a classification of the test input data.

A particular property of BCIs is that (at the time of writing) they must be fitted on a subject basis, i.e., if the system is trained using data from one user, the BCI will very likely not work properly when tested by another user. Adding to this is the fact that, to command the BCI, the subject must be imagining movement and this cannot be automatically verified (a bootstrapping problem), leading to the difficulty/impossibility of training the system by simply telling the subject to perform a specific MI.

To mitigate the problem of the initial training procedure, a trial-based approach is very commonly taken: the subject stands in front of a screen where, for each trial, a cue of an MI to perform is shown with a pre-determined temporal structure, and the subject is assumed to be performing the MI. After some sessions of training one can add feedback to these sessions which is given by running a BCI trained with the first set of data and outputting it’s classification result. An example of the time structure for trial-based BCIs is presented in Figure 2.4

![Figure 2.4](image)(a) Time structure for a trial without feedback. An arrow (the cue) appears and points at the direction which is related to MI in question.

![Figure 2.4](image)(b) Time structure for a trial with feedback. A face appears on the center of the screen, along with a bar which indicates the intended MI. The subject’s input is then interpreted and used to move the face, which in turn becomes happy as it approaches the bar and sad as it distances itself from the bar.

Figure 2.4: Examples of time structures for trials without feedback and with feedback. Source: [14]

This trial-based approach can then be carried over to the test phase, in which case the BCI is called synchronous, as opposed to BCIs which operate continuously, called asynchronous. The former usually requires that, besides distinction between classes of MI, the BCI recognizes intent of control, that is, that the user is really trying to issue orders to the BCI. Since no hardware components for real-world testing were available for this thesis, tests were run on a trial structured
2.4 State of the art

Although brain activity has been studied and analysed for a long time in a clinical context, the field of BCIs has only received increased popularity in recent years due to advances in signal acquisition equipment and increased parallel processing power in computers.

One of the first developed BCIs is the Thought Translation Device, presented in [15]. This system uses EEG to extract slow-cortical brain potentials and allows a completely paralysed user to spell into a computer at an average rate of 1 word/minute. It also employed some spelling aids like word/sentence completion and allowed visual as well as auditory feedback.

In [16], a system using visually evoked potentials allowed users to focus on a grid of letters to spell at a rate of 10-12 words/minute. For this, however, the user must have stable control of oculomotor muscles.

Another approach for spelling devices is presented in [17] and uses a specific response of the brain to stimuli which appears over the parietal cortex about 300 ms (hence its denomination of P300) after the stimuli has occurred.

With different applications in mind, the Graz [11], Albany [10] and Berlin [7] BCIs all take the MI approach and have been used for different purposes such as to control a virtual avatar [18] or navigate through Google Earth [19].

A merger of techniques from both BCI and robotics used to direct a robot is presented in [20], with the user only issuing high level commands such as “turn left at the next intersection”, thus alleviating the need for high information transfer rates in BCI.

In [5], several challenges of BCI technology are presented, namely the need for equipment which requires less setup (EEG caps, while very portable, require conductive gel to be used properly and as such still cannot be deployed in common situations) and the lack of data regarding the amount of training required before the BCI can be effectively put to use.

The BCI field suffers, in the meanwhile, from a lack of objective metrics in which to measure performance, as well as the non-existence of a common framework for standardized BCI implementation, which [21] and [22], respectively, try to address.

The CSP method is described in [3] and extended in both [12] and [13], and these are the algorithms providing a basis for the work in this thesis.
2. Basics of Brain-Computer Interfacing

Summary

In this chapter, the structure of a Brain-Computer Interface (BCI) was presented as similar to that of many common machine learning problems. Also, Event-Related Desynchronization (ERD) and Event-Related Synchronization (ERS) were presented as the observed phenomena in the Electroencephalogram (EEG) when a subject imagines performing Motor Imagery (MI). Because of the brain's lateralization, the spatial locations of ERD and ERS when performing different MI can provide an additional criterion with which to differentiate brain activity.

These notions are the basis for the methods described in the next chapter.
3

Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

Contents

3.1 Algorithms for Brain-Computer Interfaces (BCIs) ........................................ 14
3.2 Selection of Spatial Filters ............................................................................. 25
3.3 Frequency Band Selection ............................................................................. 26
3.4 Combined Frequency Band Spatial Patterns (CFBSP) ................................. 28
3.5 Spatial Patterns for Maximal Separability (SPMS) ....................................... 30
3.6 Spatial Filter Residual Analysis (SFRA) ....................................................... 32
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

This chapter presents the theoretical aspects of several methods within the scope of Common Spatial Patterns (CSP)-based Brain-Computer Interfaces (BCIs), with each section focusing on a specific method. Section 3.1 presents an overview of three already existing BCI algorithms, which deal primarily with the feature extraction steps of the BCI (see Figure 2.3). Sections 3.2, 3.3, 3.4, 3.5 and 3.6 describe methods developed for the purpose of this thesis, with Sections 3.2, 3.5 and 3.6 focusing on feature extraction while the remaining two focus on pre-processing.

A particular notice must be made to the description of the CSP method in Section 3.1.1, as this provides the basis for all the other methods in this thesis by defining both how features are computed as well as defining the concept of spatial filtering.

Implementation of the methods presented in this chapter is described in the next chapter and a comparison of their behaviour and efficacy is made in Chapter 5.

Notation The following notation is adopted in order to facilitate the reading of mathematical expressions:

- Upper case letters with no particular formatting are used to represent scalar constants. Example: “C is the number of EEG channels”;

- Upper case letters in bold refer to a matrix. Example: “\( \mathbf{W} \) is the spatial filter matrix”;

- Lower case letters in bold refer to arrays. In the case where a matrix with the same letter is used, then the lower case letter refers to a row/column of that matrix. Example: “\( \mathbf{b} \) is an eigenvector taken from the eigenvector matrix \( \mathbf{B} \)”;

- Lower case letters with no particular formatting refer to a particular element of the array/matrix with the corresponding bold/upper case bold letter. Each particular case should be clear from the indexing. Example: “\( w_{nk} \) is the \( k \)’th element in \( w_n \) and the element in the \( n \)’th row and \( k \)’th column of \( \mathbf{W} \)”;

- Functions are represented by bold letters followed by the function variables in parenthesis and the function’s value at a certain point is indicated with no bold formatting. Signals are written as functions of time and, since only sampled signals are considered in this work, \( t \) is used as the time variable instead of \( n \). Example: “\( h_k(t - 4) \) is a signal delayed by 4 samples and \( h_k(4) \) is the value of \( h_k(t) \) at \( t = 4 \)”.

3.1 Algorithms for Brain-Computer Interfaces (BCIs)

In this section, several already existing algorithms used in the context of BCIs are described. In 3.1.1, the CSP algorithm, which is used as the basis for the other algorithms, is described. This method deals with feature extraction and relies on combining EEG channels to maximize or
minimize the variance of the resulting signal using the concept of spatial filters (the term spatial
applies due to the association of each channel with an electrode which is in turn positioned at a
specific spatial location over the scalp).

CSSP (Section 3.1.2) and CSSSP (Section 3.1.3) also deal with feature extraction, and expand
the former algorithm by introducing time structure information.

3.1.1 Common Spatial Patterns (CSP)

The CSP algorithm [3] is the basis for most of the currently developed EEG-MI-BCIs. This
algorithm is a feature extraction method which attempts to enhance the effects of ERD and ERS
present in the EEG signal by computing a combination of the data from each channel in a way
that maximizes the variance of the resulting signal for one class (i.e. one kind of MI) and min-
imizes it for the other. Although it is very successful in discriminating brain states, it has the
somewhat cumbersome restriction that it can only be applied to a pair of classes at a time and
forces multiclass applications to use one-against-all and similar approaches.

A diagram of how CSP is applied is shown in Figure 3.1.

CSP works as follows: given an $E$ matrix of pre-processed EEG data with size $C \times T$ corre-
sponding to $T$ samples taken from each of the $C$ EEG channels available,

$$E = \begin{bmatrix} E_1(t) \\ E_2(t) \\ \vdots \\ E_C(t) \end{bmatrix}$$  \hspace{1cm} (3.1)

where $E_n(t) = [E_n(0) \hspace{0.2cm} E_n(1) \hspace{0.2cm} \cdots \hspace{0.2cm} E_n(T-1)]$, we will attempt to estimate a $C$ length array, $w$, such that the signal $z$ resulting from the projection of $E$ onto $w$ has maximum variance if $E$ is from
a trial in class 1 and minimum variance if it is from class 2, i.e.,

$$z = w^T E,$$  \hspace{1cm} (3.2)

$E_{\text{class}1} \Rightarrow \text{var}(z) \uparrow,$  \hspace{1cm} (3.3)

$E_{\text{class}2} \Rightarrow \text{var}(z) \downarrow.$  \hspace{1cm} (3.4)

In this simple case, $w$ is a single array which leads us to compute a single feature from the
input data. This can be easily extended to $N$ dimensions by using
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

Pre-Processing

CSP

Spatial Filtering
And log-
normalization of
Variance

Classifier Training

To Testing...

EEG Data

Pre-Processed
EEG Data (E)

Spatial Filters (W)

Features (f)

Classifier

(a) Training procedure for the CSP algorithm.

EEG Data

Pre-Processing

Pre-Processed
EEG Data (E)

Spatial Filtering
And log-
normalization of variance

Classification

Features (f)

Labels

Spatial Filters (W)

Classifier

(b) Testing procedure for the CSP algorithm.

Figure 3.1: Diagram for the training and testing phases of CSP.
3.1 Algorithms for Brain-Computer Interfaces (BCIs)

\[ Z = W^T E, \]  
\[ W^T = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_N^T \end{bmatrix}, \]  
\[ Z = \begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_N(t) \end{bmatrix}. \]

Note that, if \( W \) is a \( C \times N \) matrix, then \( Z \) is \( N \times T \). We compute the features for classification by taking the variance of each of the rows of \( Z \), thus obtaining an \( N \)-length feature vector \( g \) which is then normalized and log-transformed so that every feature in the new feature vector, \( f \), will have a gaussian distribution \([2]\).

\[ g = \begin{bmatrix} \text{var} (z_1(t)) \\ \text{var} (z_2(t)) \\ \vdots \\ \text{var} (z_N(t)) \end{bmatrix}, \]
\[ f = \log \left( \frac{g}{|g|_1} \right). \]

Notice that, if we are employing CSP with a single feature, the normalization step must be skipped or the resulting feature will always be 0.

As a simple example, consider the EEG signals presented in Figure 3.2.

Figure 3.2: Example EEG signals for right hand and foot MIs. The legend is the value of the signals variance.

For these particular signals, we could devise a simple spatial filter for distinguishing the two types of motor imagery by multiplying the upper channel by 1 and the lower channel by 0. In this
manner, we retain only the upper channels, and by checking the variances, we can see that right hand imagery causes much higher values of variance to appear than foot imagery. Supposing we wanted two filters, we could design a second filter to perform the converse operation of the first one. It should be noted that this is clearly not a reliable method for computing spatial filters, both because we developed the filters by looking at a single trial from each class (and we have no guarantee that either of these trials is a good representative of trials for their class) and because the number of electrodes (channels) of an EEG acquisition system can vary between less than 10 and over 100, and, in the latter case, it is clearly not practical to estimate filters by hand.

Now that all of the steps for feature extraction in the CSP context have been explained, all that’s left is to outline the method through which CSP estimates the $W$ matrix.

This method will, however, be easier to understand if we first attempt to express the CSP problem as a general optimization problem. We define it for the simple case in which there is a single filter $w$:

$$w = \arg \max_x \quad \text{var}(x^T E_k)$$

s.t. \quad \text{var}(x^T E_1) + \text{var}(x^T E_2) = 1, \quad k = \{1, 2\}. \quad (3.10)$$

It is clear from this formulation that we are attempting to maximize variance for class $k$ and, by inspecting the restriction, separating the classes: since the class variances add up to 1 and we are maximizing one of them, we are simultaneously minimizing the other. Another interesting fact about this restriction is that it makes the problem symmetric in terms of classes and thus the choice of $k$ in the cost function is purely aesthetic.

By using the properties of the variance, we change (3.10) to:

$$w = \arg \max_x \quad x^T \Sigma_k x$$

s.t. \quad $x^T \Sigma_1 x + x^T \Sigma_2 x = 1, \quad k = \{1, 2\}$,

where $\Sigma_k$ is the covariance matrix of $E_k$.

With this formulation, we can finally describe how CSP estimates the spatial filters. First, we compute the covariance matrices $Y_n$ for each trial data $E_n$ (in this case the index is used to refer to the trial rather than class):

$$Y_n = \frac{E_n E_n^T}{tr(E_n E_n^T)}.$$ \hspace{1cm} (3.12)

Then, we obtain two average covariance matrices, one for each class, by averaging across all trials of that class:

$$Y_k = \frac{\sum_{n=1}^{N_k} Y_{nk}}{N_k}, \quad k = \{1, 2\}. \quad (3.13)$$

$Y_{nk}$ is the covariance matrix of trial $n$ for class $k$ and $N_k$ is the number of trials for class $k$. 

3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)
3.1 Algorithms for Brain-Computer Interfaces (BCIs)

We now define a composite covariance matrix obtained from adding the average covariance matrices for both classes,

\[ Y_c = \bar{Y}_1 + \bar{Y}_2. \]  

(3.14)

This composite covariance matrix can be eigendecomposed so that

\[ Y_c = U_c \lambda_c U_c^T, \]

(3.15)

where \( U_c \) is a matrix with the eigenvectors of \( Y_c \) and \( \lambda_c \) a diagonal matrix with the respective eigenvalues.

The variances in the space spanned by \( U_c \) can be whitened by

\[ P = \sqrt{\lambda_c^{-1}} U_c^T \]

(3.16)

and by computing

\[ S_k = P Y_k P^T, k = \{1, 2\} \]

(3.17)

we ensure that \( S_1 \) and \( S_2 \) have the same eigenvector matrix, \( B \), and that the sum of the two eigenvalues for each eigenvector, \( \lambda_1 + \lambda_2 \), is 1, because

\[ P Y_1 P^T + P Y_2 P^T = P (\bar{Y}_1 + \bar{Y}_2) P^T = P Y_c P^T = I, \]

(3.18)

where \( I \) is the identity matrix. Because we’re dealing with covariance matrices which are, therefore, definite positive, we have the added restriction that \( \lambda_k \in [0, 1] \).

We thus obtain

\[ W = B^T P, \]

(3.19)

where \( W \) is employed as in (3.5).

Why this method provides a solution for (3.10) should be clear after understanding that

\[ \text{var}(w^T E) = w^T \Sigma w = b^T P \Sigma P^T b = b^T \lambda_k b = \lambda_k \langle b, b \rangle, \]

(3.20)

where \( \Sigma \) is the covariance matrix of \( E \) and \( \lambda_k \) is the eigenvalue associated with \( b \) when \( E \) is data from a trial in class \( k \). Because \( b \) is an eigenvector of \( P \Sigma P^T \), \( P \Sigma P^T b \) simplifies to \( \lambda b \) and thus the result is the norm of \( b \) scaled by the eigenvalue associated with it.

Recalling (3.18), we conclude that, with the exception of the case where \( \lambda_{1,2} = 1/2 \), we obtain different values for trials in different classes and can thus separate them. Considering the case where \( \lambda_1 = 3/4 \Leftrightarrow \lambda_2 = 1/4 \), we would then have \( \text{var} (w^T E_1) = \frac{3}{4} \langle b, b \rangle \) and \( \text{var} (w^T E_2) = \frac{1}{4} \langle b, b \rangle \) where \( E_1 \) and \( E_2 \) are data matrices from trials in class 1 and 2, respectively.
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

The \( W \) matrix obtained from the above steps should not, however, be used directly to run the algorithm. The first reason for this is that it has the same number of filters as the number of channels, with each filter producing a component of the feature vector. Thus, using all of the filters in systems with many channels will adversely affect the computational performance of the BCI, as not only feature extraction is more intensive but classification time also increases with an increase in the number of features (and generally this increase is superlinear in relation to the number of features). Secondly, not all filters perform well (even in theoretical terms, cases where \( \lambda = 1/2 \) are useless) and can confuse the classifier and as such it is advisable to remove the poorly performing filters. From (3.20), it becomes clear that filter selection is closely related to the values of eigenvalues. A more in-depth approach to how filter selection should be performed is presented in Section 3.2.

3.1.2 Common Spatio-Spectral Patterns (CSSP)

The Common Spatio-Spectral Patterns (CSSP) algorithm \([12]\) attempts to expand the CSP algorithm by adding some information about the temporal structure of the EEG signals. This is done by appending delayed versions of the original EEG signal's channels as new channels (shown in Figure 3.3, i.e., we still formulate the problem as in (3.2) but our input data matrix, \( E \), becomes

\[
E_{\text{CSSP}} = \begin{bmatrix}
E_{\text{CSP}} (t) \\
E_{\text{CSP}} (t - \tau_1) \\
E_{\text{CSP}} (t - \tau_2) \\
\vdots \\
E_{\text{CSP}} (t - \tau_N)
\end{bmatrix},
\]

(3.21)

where \( E_{\text{CSP}} (t - \tau_n) \) is the signal as used in CSP (the signal with \( C \) channels obtained from the EEG equipment) delayed by \( \tau_n \) samples. Hence, if \( N \) delayed versions are appended, \( E_{\text{CSSP}} \) will have size \( NC \times T \) (recall that \( T \) is the number of samples obtained per trial).

To provide a more formal insight into the output of the algorithm, we begin by applying a single filter from the whole set obtained from running the algorithm (since every filter is applied in the same way and is independent from other filters, this will not affect the conclusion):
3.1 Algorithms for Brain-Computer Interfaces (BCIs)

Pre-Processing

CSP

Spatial Filtering

And log-normalization of Variance

Classifier Training

EEG Data

Pre-Processed EEG Data (ECSP)

CSP

Spatial Filters (W)

Features (f)

Classifier Training

Delaying

Concatenation

To Testing...

EEG Data

Pre-Processing

Delaying

Concatenation

Features (f)

Classifier

Pre-Processed EEG Data (ECSP)

Spatial Filtering

And log-normalization of Variance

Classification

Labels

From Training...

Figure 3.3: Diagram for the training and testing phases of CSSP.
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

\[
z = w^T E_{\text{CSP}} = \begin{bmatrix} w_0^T & w_1^T & w_2^T & \cdots & w_N^T \end{bmatrix}
\]

\[
\begin{bmatrix}
E_{\text{CSP}_1}(t) \\
E_{\text{CSP}_2}(t) \\
\vdots \\
E_{\text{CSP}_C}(t) \\
E_{\text{CSP}_1}(t-\tau_1) \\
E_{\text{CSP}_2}(t-\tau_1) \\
\vdots \\
E_{\text{CSP}_C}(t-\tau_1) \\
E_{\text{CSP}_1}(t-\tau_2) \\
E_{\text{CSP}_2}(t-\tau_2) \\
\vdots \\
E_{\text{CSP}_C}(t-\tau_2) \\
\vdots \\
E_{\text{CSP}_1}(t-\tau_N) \\
E_{\text{CSP}_2}(t-\tau_N) \\
\vdots \\
E_{\text{CSP}_C}(t-\tau_N)
\end{bmatrix}
\]

(3.22)

where \( w_n^T = [w_{n1} \ w_{n2} \ \cdots \ w_{nC}] \). The representation of \( w \) as a concatenation of \( w_n \)'s is used both to facilitate the comprehension of the next steps and to provide the intuition that the big filter \( w \) is in fact the result of joining a set of filters which perform spatial filtering on each copy of the original signal.

Rewriting (3.22) as a sum yields

\[
z = \sum_{n=0}^{N} w_n^T E_{\text{CSP}}(t - \tau_n), \quad \tau_0 = 0,
\]

(3.23)

which can be further unrolled as

\[
z = \sum_{n=0}^{N} \sum_{k=1}^{C} w_{nk} E_{\text{CSP}_k}(t - \tau_n), \quad \tau_0 = 0.
\]

(3.24)

By exchanging the order of summation,

\[
z = \sum_{k=1}^{C} \sum_{n=0}^{N} w_{nk} E_{\text{CSP}_k}(t - \tau_n), \quad \tau_0 = 0,
\]

(3.25)

we can identify the inner sum as a filtering operation, that is, for every channel \( k \), we can define a max \( \tau_n + 1 \) length FIR filter with impulse response \( h_k(t) \) which has each \( \tau_n \)'th coefficient equal to \( w_{nk} \) for \( n \in [0, N] \) and every other coefficient set to 0,

\[
z = \sum_{k=1}^{C} h_k(t) * E_{\text{CSP}_k}(t).
\]

(3.26)

We also notice that the spatial filtering is now implicit in the spectral filtering, as the channels are summed after filtering without weighting. As an example, suppose we appended only one
copy of the original EEG data, delayed by 3 samples. We would then have, for an input data with
2 channels:

\[ \begin{align*}
\tau_1 = 3 & \Rightarrow h_1(t) = \begin{bmatrix} w_{01} & 0 & 0 & w_{11} \\ w_{02} & 0 & 0 & w_{12} \end{bmatrix}, \\
h_2(t) & = \begin{bmatrix} w_{01} & 0 & 0 & w_{11} \\ w_{02} & 0 & 0 & w_{12} \end{bmatrix}.
\end{align*} \tag{3.27} \]

Because we originally have two channels, CSSP will output two filters, \( h_1(t) \) and \( h_2(t) \), one for each channel. The 3 samples delay will cause the filters to be length \( 3 + 1 = 4 \) and have every coefficient equal to zero except for the 0'th and the third coefficients.

Note, however, that the above is one possible interpretation and, in practice, the algorithm can be run by applying \( (3.22) \) as well.

### 3.1.3 Common Sparse Spectral Spatial Patterns (CSSSP)

Common Sparse Spectral Spatial Patterns (CSSSP) \[13\] is another variation of CSP which simultaneously optimizes a spatial filter and a spectral filter by using numerical optimization. Because a spectral filter is computed in addition to the spatial filters, \( (3.10) \) changes to

\[
\begin{align*}
\mathbf{w} & = \arg_{\mathbf{x}, \mathbf{b}} \max \quad \text{var} \left( \mathbf{x}^T \mathbf{E}_k \right) \\
\text{s.t.} & \quad \text{var} \left( \mathbf{x}^T \mathbf{E}_1 \right) + \text{var} \left( \mathbf{x}^T \mathbf{E}_2 \right) = 1, \quad k = \{1, 2\},
\end{align*} \tag{3.28} \]

where \( \mathbf{E}_k \) is the data from a trial that has been filtered using the spectral filter \( \mathbf{b} \).

A diagram of how CSSSP is applied is shown in Figure 3.4. Note that, contrary to the CSSP algorithm, the spectral filtering is now explicitly applied.

Defining the spectral filter to be causal, have finite impulse response, length \( L \) and coefficients \( b(k) \) with \( b(1) = 1 \), we can represent the filtered input data as

\[
\mathbf{E}_{\text{CSSSP}} = \mathbf{E}_{\text{CSP}}(t) + \sum_{\tau=2}^{L} b(\tau) \mathbf{E}_{\text{CSP}}(t-\tau). \tag{3.29} \]

Notice that the filtering is now the same for all the channels, as opposed to the implicit filtering for each channel in CSSP.

Substituting \( (3.29) \) into \( (3.10) \) and performing some approximations will yield

\[
\mathbf{b} = \arg_{\mathbf{b}, \mathbf{x}} \max \quad \mathbf{x}^T \mathbf{A}_k \mathbf{x} \\
\text{s.t.} \quad \mathbf{x}^T \mathbf{A}_1 \mathbf{x} + \mathbf{x}^T \mathbf{A}_2 \mathbf{x} = 1, \quad k = \{1, 2\},
\]

where \( \mathbf{A}_k = \sum_{\tau=0}^{L-1} \sum_{j=1}^{L-\tau} b(j) b(j+\tau) \mathbf{1}_k \) and \( \mathbf{1}_k = \frac{\mathbf{E}_{\text{CSP}}(t) \mathbf{E}_{\text{CSP}}(t-\tau)^T}{\text{tr} \left( \mathbf{E}_{\text{CSP}}(t) \mathbf{E}_{\text{CSP}}(t-\tau)^T \right)} \).

The optimization process alternates between optimizing the spectral filter using common numerical optimization algorithms like the simplex method and line search, and computing the corresponding spatial filter using the CSP method.

CSSSP also requires a parameter \( R \), which is a non-negative regularization constant used for obtaining sparse solutions and avoid overfitting, usually chosen by cross-validation.
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

Figure 3.4: Diagram for the training and testing phases of CSSSP.
3.2 Selection of Spatial Filters

As stated in Section 3.1.1, by selecting only a few columns of the spatial filters matrix $W$ obtained by CSP, one can reduce the number of dimensions for the machine learning task at hand. By inspecting the results from (3.20) and (3.18), one can conclude that spatial filters associated with the highest and lowest eigenvalues should be picked.

The way filters are typically chosen is by declaring a positive and even integer $m$ and taking the $m/2$ spatial filters with highest eigenvalue and the $m/2$ spatial filters with lowest eigenvalue. This approach has been named *traditional*. The value of $m$ is usually either chosen from the range $[4; 16]$ (the choice, as well as the range itself, stems from previous empirical evidence and is based on personal taste) or by testing various values of the parameter on the training data and choosing the one which performs best.

Surprisingly, little and/or unsatisfying justification is usually found to why spatial filter selection is performed in this manner, especially if one considers a possible and important caveat regarding the procedure. Suppose our input training data is acquired with 8 channels. Running the CSP method in this training data would yield 8 spatial filters. Suppose now that we had the following eigenvalues associated with these filters:

$$
\lambda = \begin{bmatrix}
1 & 5 & 3 & 3 & 1 & 2 & 1 & 0 \\
\end{bmatrix}.
$$

We would, for example, set $m = 4$ and thus choose the spatial filters associated with eigenvalues $\{1, 5, 3, 1\}$. Although, in an ideal world, this wouldn’t pose a problem as our ideal classifier would just discard the features produced by the filter associated with the eigenvalue $1\over2$, in real applications, that particular filter could confuse the classifier.

This caveat is especially perilous if we add the fact that the covariance matrix for the trial we are classifying will probably be different from the covariance matrix we used for computing the filters and thus have different eigenvalues even after the whitening procedure! Formally,

$$
\Sigma_r = \Sigma_t + \Sigma_e,
$$

where the real covariance $\Sigma_r$ is represented as a sum of a theoretical covariance matrix $\Sigma_t$ (which corresponds to the theoretical covariance matrix of one of the classes of the problem) and an “error” matrix $\Sigma_e$.

We the apply this decomposition of $\Sigma_r$ in (3.20) to obtain

$$
\text{var}(W^T E) = \lambda_r \langle b, b \rangle + b^T P \Sigma_e P^T b,
$$

where it is clear that the distribution of the feature becomes dependent on the distribution of $b^T P \Sigma_e P^T b$. Returning to our concrete example above, if the $\lambda = 1\over2$ filter is applied, then the term $\lambda_r \langle b, b \rangle$ has the same value for trials in both classes and, unless $b^T P \Sigma_e P^T b$ has a magical
distribution which still allows the classes to be separable (it doesn’t), the feature will probably lead the classifier in the wrong direction.

This motivates the development of different ways of selecting spatial filters. Here, two methods are presented, relying solely on analysis of a new parameter $\Lambda = \max\{\lambda_1, 1 - \lambda_2\}$ which has been called the separability:

- the *sorted* method, where one chooses the $m$ filters with largest $\Lambda$, i.e., for each spatial filter $w_k$ with associated eigenvalues $\lambda_{k1}, \lambda_{k2}$, we compute $\Lambda_k = \max\{\lambda_{k1}, 1 - \lambda_{k2}\}$ and select the $m$ filters for which the obtained value is highest;

- the *mixed* method, where one chooses the filter with highest $\lambda_1$ and the one with the highest $\lambda_2$, but proceeds to select $m - 2$ additional filters following the order of largest $\Lambda$.

The *mixed* method might not make much sense at first, but was developed to address a possible caveat of the *sorted* which is the normalization of the feature vector, as it could happen that all the selected eigenvalues are very close to each other and thus, after normalizing, the distributions of the feature vector for each class would overlap. Another trivial property of the *mixed* method is that, for $m = 2$, it is the same as the *traditional* approach. Finally, the conclusion can be drawn that, under certain conditions, all the methods end up choosing the same filters.

### 3.3 Frequency Band Selection

Before running the CSP algorithm or one of its variants, the input data is usually put through a pre-processing stage. This stage is usually comprised of filtering in the frequency domain and/or some other signal processing such as Independent Component Analysis or Regression which serve the purpose of denoising the input signal and removing some particular artifacts (such as the ones related to movement of the subject’s eyes).

Although it is generally agreed that BCI performance is greatly influenced by the input signal’s quality, most approaches deal with the feature extraction phases and neglect the pre-processing. In particular, many times the filtering employed is a mere bandpass filter with cutoff frequencies around 5 and 30 Hz, and, while this choice performs reasonably well, it leaves a great margin for improvement.

In order to improve the accuracy with which the BCI predicts mental activity, a wrapper approach frequency search method was developed. An initial set of bands is generated by defining a frequency range, the minimum width of each band and the frequency granularity (the distance between the central frequencies of two contiguous bands). An example of an initial set would be obtained by breaking up the $[4 \rightarrow 20]$ Hz range into 4 Hz wide bands with central frequencies $\{6, 10, 14, 18\}$ Hz.

The algorithm begins with a first round of selection using this initial set of bands. The procedure for each band is the same as when normally running the CSP algorithm: the data is filtered
3.3 Frequency Band Selection

and used to train the classifier. Each band is then scored with the accuracy of the trained classifier on the training data, which is why the approach is said to be of the wrapper type (because the scoring is wrapped around the classifier). The winning band in this round is then used to construct a second set of bands. This second set is composed of the winning band and the bands obtained by stretching one of the winning band’s bounds by a multiple of the granularity. Band generation is bounded to bands which are in the range specified initially. Another round of testing is then performed in similar fashion to the first and the winner of this second round is used to design the pre-processing filter.

The complexity of this algorithm can be better understood if one resorts to a tree representation, with each node representing a band. From the formulation of our problem, we also define $G$, $R$ and $W$, the frequency granularity, range and minimum bandwidth, respectively. The tree’s root is the band which fills the entire range defined initially and it’s children are the bands obtained by subtracting the granularity from one of the bounds. An abstract example of a tree is shown in Figure 3.5(a), where $xG - R$ and $R - xG$ are used to represent bands from which $G$ has been subtracted $x$ times from the lower and upper bound, respectively. Figure 3.5(b) shows a concrete tree example using the initial bands from above.

![Tree Diagram](image)

(a) Abstract tree for band set generation.  
(b) Example of a tree built with range 4-20 and both granularity and minimum bandwidth 4.

Figure 3.5: Trees built for band selection.

We can immediately conclude that, since, at each level of the tree, bands have less $G$ bandwidth than the ones in the level above, the tree will have $\frac{R-W}{G} + 1$ levels, rounded down. Additionally, each level has the number of nodes from the above level plus one. To prove this, we will say that the root’s level is 0. We can then use the notation introduced above to describe every node in level $v$, $xG - R - Gy$, where it is easy to see that $x + y = v$. Because $x$ can only take integer values in the range $[0, v]$ and $v$ increases by one each time we go down one level, we conclude that $v + 1 \Rightarrow x \in [0, v + 1]$ and thus there is one more node in level $v + 1$ than there is in level $v$. 

27
This results in that the \( v^{th} \) level has \( v + 1 \) nodes and the tree will have
\[
\frac{R - W}{G} \sum_{n=0}^{n+1} n + 1 = \left( \frac{R - W}{G} \right)^2 + \frac{R - W}{G} + 2
\]

nodes, which is the complexity of a full search on the tree.

The proposed algorithm still has a complexity which is quadratically related to the number of bands in the initial set. However, from the tree interpretation, one can thus see that the first round of selection uses only the leaves of the tree while the second round searches only the nodes of the tree which represent bands containing the winner of the first round (see Figure 3.6), thus reducing the cost of the search.

![Diagram of band selection](image)

(a) State of the search after the first round.  
(b) State of the search after the second round if the 16 – 20 Hz band won the first round.

(c) State of the search after the second round if the 4 – 8 Hz band won the first round.  
(d) State of the search after the second round if the 12 – 16 Hz band won the first round.

Figure 3.6: An example of running band selection. Nodes searched in the first round are highlighted in light grey and nodes searched in the second round are highlighted in dark grey.

### 3.4 Combined Frequency Band Spatial Patterns (CFBSP)

Motivated by the algorithms described in Sections 3.1.2 and 3.3, a new CSP-based algorithm called Combined Frequency Band Spatial Patterns (CFBSP) was developed. This new method acknowledges the fact that ERD and ERS are spectrally and spatially defined and, as such, it makes sense to combine both sources of information.
3.4 Combined Frequency Band Spatial Patterns (CFBSP)

Figure 3.7 presents a schematic view of the CFBSP.

While the previous algorithms simply use spectral information to increase the signal-to-noise ratio by tightening a spectral filter, the CFBSP algorithm can process several frequency bands simultaneously and merge the information from each of those bands.

This is achieved by creating a new data matrix

$$E_{CFBSP} = \begin{bmatrix} E_{CSP_1} \\ E_{CSP_2} \\ \vdots \\ E_{CSP_N} \end{bmatrix},$$

where each $E_{CSP_n}$ is the input data of the CSP algorithm filtered into one of $N$ spectral filters.
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

$E_{CFBSP}$ is then used as normally in the CSP method to obtain filters which combine spatial and spectral information.

To select which frequency bands to filter the signals into, a simplified version of the algorithm described in Section 3.3 which consists of defining a variable $\nu$, running the first round of band selection and selecting the $\nu$ bands which perform best.

3.5 Spatial Patterns for Maximal Separability (SPMS)

The SPMS algorithm attempts to improve classification by computing spatial filters which separate the distributions of features obtained by CSP for each of the classes. This is motivated by the fact that, rather than computing spatial filters and then selecting the best ones, which can sometimes lead to an overall poor choice of filters, it can be better, accuracy-wise, to define how many spatial filters to compute and ensure that every one of those filters performs reasonably well. Thus the SPMS algorithm simply redefines the spatial filter computation portion of the CSP algorithm, as can be seen in Figure 3.8.

![Diagram for the training and testing phases of SPMS.](image)

Figure 3.8: Diagram for the training and testing phases of SPMS.
3.5 Spatial Patterns for Maximal Separability (SPMS)

It does this by assuming that the \( F \) length feature vector \( x \) is gaussian distributed for each of the classes,

\[
p(x|k) = \frac{1}{(2\pi)^{F/2} |\Sigma_k|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right\},
\]

(3.35)

where \( k \in \{1, 2\} \) is the class label and \( \mu_k \) and \( \Sigma_k \) are the mean vector and covariance matrix for class label \( k \), respectively.

The classes can then be separated by minimizing their distribution overlap, i.e.,

\[
\int_{\mathbb{R}^F} p(x|1) p(x|2) \, dx.
\]

(3.36)

Since we have assumed gaussianity, (3.36) turns into

\[
\int_{\mathbb{R}^F} \frac{1}{(2\pi)^{F/2} |\Sigma_1|^{1/2} |\Sigma_2|^{1/2}} \left( -\frac{1}{2} (x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) - \frac{1}{2} (x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2) \right) \, dx.
\]

(3.37)

Manipulation of the exponential part yields

\[
(x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) + (x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2) = (x - \mu_12)^T \Sigma_{12}^{-1} (x - \mu_1) - \mu_{12}^T \Sigma_{12}^{-1} \mu_2 + \mu_1^T \Sigma_{12}^{-1} \mu_1 + \mu_2^T \Sigma_{12} \mu_2,
\]

(3.38)

where \( \Sigma_{12}^{-1} = \Sigma_1^{-1} + \Sigma_2^{-1} \) and \( \mu_{12} = \Sigma_1 \mu_1 + \Sigma_2 \mu_2 \).

Joining (3.37) and (3.38) results in:

\[
\int_{\mathbb{R}^F} p(x|1) p(x|2) \, dx =
\]

\[
\frac{|\Sigma_{12}|^{1/2}}{(2\pi)^{F/2} |\Sigma_1|^{1/2} |\Sigma_2|^{1/2}} e^{(-\mu_{12}^T \Sigma_{12}^{-1} \mu_{12} + \mu_1^T \Sigma_{12}^{-1} \mu_1 + \mu_2^T \Sigma_{12} \mu_2)} \int_{\mathbb{R}^F} \frac{1}{(2\pi)^{F/2} |\Sigma_{12}|^{1/2}} e^{-(x - \mu_{12})^T \Sigma_{12}^{-1} (x - \mu_{12})} \, dx =
\]

\[
\frac{|\Sigma_{12}|^{1/2}}{(2\pi)^{F/2} |\Sigma_1|^{1/2} |\Sigma_2|^{1/2}} e^{(-\mu_{12}^T \Sigma_{12}^{-1} \mu_{12} + \mu_1^T \Sigma_1^{-1} \mu_1 + \mu_2^T \Sigma_2 \mu_2)},
\]

(3.39)

where \( |\Sigma_{12}|^{1/2} = \frac{1}{|\Sigma_1|^{1/2} |\Sigma_2|^{1/2}} \), and the last step is made by noting that the expression inside the integral is the expression for a multivariate gaussian distribution and thus integrates to 1.

Applying a logarithm to (3.39) allows us to compute the objective function’s value simply from the mean and covariance matrix of the feature vectors in the training data and thus use numerical optimization methods such as the simplex method to reach a solution, which is a spatial filter used as in CSP.
3.6 Spatial Filter Residual Analysis (SFRA)

The SFRA algorithm is yet another feature extraction method which is based on CSP. The algorithm uses the normal CSP procedure to extract spatial filters and then generates AR models. These AR models are then used during the feature extraction to filter the signals obtained after spatial filtering, thus obtaining a set of residual signals. Just like in CSP, features are then obtained by computing the variance of the residual signals and log-normalizing.

A diagram of how SFRA works is presented in Figure 3.9.

The motivation for the creation of this algorithm is adding temporal information to the basic CSP algorithm using the AR models. The aim of an AR model is to, given a signal $s(t)$, estimate a sequence $a(n)$ with length $L_a$ such that, for

$$s(t) = \sum_{n=1}^{L_a} a(n)s(t-n) + \xi(t), \quad \text{(3.40)}$$

the residual, $\xi(t)$, is gaussian distributed with zero mean.

The first term on the right side of (3.40) can be interpreted as filtering the signal with an all-pole filter. Manipulating (3.40) yields

$$s(t) - \sum_{n=1}^{L_a} a(n)s(t-n) = \xi(t), \quad \text{(3.41)}$$

which leads to the conclusion that, if $s(t)$ is filtered with $[1 \quad -a(n)]$, the resulting signal is the residual. By choosing the criterion for estimating $a(n)$ to be the mean-square prediction error, i.e.,

$$E \left[ \left( s(t) - \sum_{n=1}^{L_a} a(n)s(t-n) \right)^2 \right],$$

we are thus minimizing the power/variance of the residual signal.

It is then expected that, if an AR model is trained on a trial from one class, using it to filter trials from that class will yield a low power signal. Conversely, filtering trials from other classes will result in higher power signals because the model is unable to properly predict the temporal structure.

In the CSP context one can train a model for each spatial filter and each class. Care must be taken however, since spatial filtering is applied with a similar criterion to the model parameter estimation criterion, which is to minimize variance for one class and maximize for the other. It makes no sense, then, to extract an AR model from trials of the class for which the spatial filter maximizes the variance, since filtering trials from that class will then reduce the variance and thus perform the opposite operation of the spatial filtering. The conclusion is that a single model should be generated for each spatial filter using trials for the class whose variance is minimized.
3.6 Spatial Filter Residual Analysis (SFRA)

(a) Training procedure for the SFRA algorithm.

(b) Testing procedure for the SFRA algorithm.

Figure 3.9: Diagram for the training and testing phases of SFRA.
3. Methods for Common Spatial Patterns (CSP)-Brain-Computer Interfaces (BCIs)

Summary

This chapter presented Common Spatial Patterns (CSP), Common Spatio-Spectral Patterns (CSSP) and Common Sparse Spectral Spatial Patterns (CSSSP), three existing algorithms for Brain-Computer Interface (BCI) applications. Some methods are also proposed with the purpose of increasing the performance of these BCI algorithms by changing the way that spatial filters are selected and optimizing the pre-processing filter. Additionally, three new methods for BCI applications, which have been named Combined Frequency Band Spatial Patterns (CFBSP), Spatial Patterns for Maximal Separability (SPMS) and Spatial Filter Residual Analysis (SFRA) are proposed. CFBSP extends the basic CSP algorithm to fuse spatial and spectral data, while SPMS minimizes the overlap of the distributions of features and SFRA makes use of AR models to model the time structure of the spatially filtered signals. The implementation of all of these methods in a MATLAB environment is presented in the next chapter.
Contents

4.1 Core functions ................................................................. 36
4.2 Algorithm Implementations ............................................. 40
In the scope of this thesis, a collection of MATLAB functions was developed to allow quick prototyping and testing of new Brain-Computer Interface (BCI) algorithms. The core of the library is a skeleton function which defines a normalized structure for running algorithm simulations and a few utility functions which provide implementation for common operations in Common Spatial Patterns (CSP)-based BCI. These are described in Section 4.1.

Section 4.2 describes the implementations of the algorithms described in Chapter 3.

4.1 Core functions

4.1.1 Simulator skeleton

The skeleton function simply calls other functions and makes very few assumptions over the data model of the simulator in order to be as generic as possible. This function's structure is presented by the simplified code in Figure 4.1.

```matlab
function results = skeleton(file,varargin){
    /* Argument Parsing */
    params = parse_args(varargin);
    /* Data loading and structuring */
    [data,params] = read_data(file,params);
    /* Setup crossvalidation */
    params = setup_xval(data,params);
    /* Global Preprocessing */
    [data,params] = global_prep(data,params);
    /* Crossvalidation Cycle */
    for(/* Each crossvalidation set */){
        trainData = /* part of the full data for training */
        testData = /* part of the full data for testing */
        /* Local Preprocessing */
        [trainData,testData,runParams] = local_prep(train,test,params);
        /* Extract Algorithm Parameters */
        runParams = get_params(train,runParams);
        /* Extract Features */
        [trainFeatures,testFeatures] = get_features(train,test,runParams);
        /* Run classifier */
        runParams = run_classifier(trainFeatures,testFeatures,runParams);
        /* Gather data from crossvalidation run */
        params.RunResults = get_run_results(runParams);
    }
    /* Gather data to output */
    results = package_results(params);
}
```

Figure 4.1: Simplified code of the BCI skeleton function.

This skeleton function merely makes calls to other functions which can be rewritten by a develop-
4.1 Core functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse_args</td>
<td>Parse arguments and setup simulation parameters.</td>
</tr>
<tr>
<td>read_data</td>
<td>Read and format the simulation input data.</td>
</tr>
<tr>
<td>setup_xval</td>
<td>Set up the crossvalidation indices.</td>
</tr>
<tr>
<td>global_prep</td>
<td>Do pre-processing of the input data which is independent from it.</td>
</tr>
<tr>
<td>local_prep</td>
<td>Do pre-processing of the input data which requires training.</td>
</tr>
<tr>
<td>get_params</td>
<td>Compute required parameters for feature extraction.</td>
</tr>
<tr>
<td>get_features</td>
<td>Perform feature extraction.</td>
</tr>
<tr>
<td>run_classifier</td>
<td>Train and test classifiers on the previously extracted features.</td>
</tr>
<tr>
<td>get_run_results</td>
<td>Store important values from a single crossvalidation run.</td>
</tr>
<tr>
<td>package_results</td>
<td>Store important values which describe the entire simulation.</td>
</tr>
</tbody>
</table>

Table 4.1: Functions called by the skeleton function that can be overridden, together with their intended use.

Table 4.1 contains a listing of all the functions called by the skeleton function that can be overridden and a small description of their intended use.

To ensure maximum compatibility across systems, this skeleton function uses only basic MATLAB functions and can be used in any MATLAB installation.

The function has two inputs, file and varargin. While the first is a cell array used to store the names of files containing data to be read, the second stores in a cell array all the other arguments supplied when calling the skeleton function, with each argument occupying an element of the array. It has a single output, results, which is intended to be a structure with the simulation results. This output will be explained in better detail in the implementations in Section 4.2.

4.1.2 EEG_COV function

The eeg_cov function provides a computationally efficient way to compute the average covariance matrix for a set of trial structured EEG data. This function’s simplified code is presented in Figure 4.2.

This function computes the covariance of each trial using (3.12) and circumvents the MATLAB built-in function cov to increase performance. It assumes that the input data is a cell array in which each element is a $T \times S \times C$ 3-D array with EEG data from one class, where $T$, $S$ and $C$ are the number of trials, number of samples and number of channels, respectively. The output averageCovMat is a cell array with each element containing the $C \times C$ average covariance matrix of the class in the corresponding element of data.
function averageCovMat = eeg_cov(data){
    /* zeros(n) allocates a matrix filled with zeros that has n rows and n columns */
    covMats = zeros(numTrials*numChannels*numChannels);

    for /* each class in data */) {
        for /* k running each trial in data */) {
            trialData = data(k);
            covMats(k,:,:) = trialData*transpose(trialData);
            covMats(k,:,:) = covMats(k,:,:)/trace(covMats(k,:,:));
        }
    }
    averageCovMat = mean(covMats);
}

Figure 4.2: Simplified code for the eeg_cov function.

4.1.3 EEG_CSP function

The eeg_csp function implements the CSP method of obtaining spatial filters described in Section 3.1.1.

The simplified code for this function is presented in Figure 4.3.

function [W,D] = eeg_csp(meanCovMat) {
    Cc = meanCovMat(1) + meanCovMat(2);
    /* Here eig performs eigendecomposition on Cc. 
       U is the eigenvector matrix with the eigenvectors in the columns. 
       lambda is a diagonal matrix with the corresponding eigenvalues. */
    [U,lambda] = eig(Cc);
    P = sqrt(invert(lambda))*transpose(U);
    Sl = P*meanCovMat(1)*transpose(P);
    Sr = P*meanCovMat(2)*transpose(P);
    /* Now eig performs eigendecomposition on Sr and Sl which have 
       the same eigenvectors. 
       D is a diagonal matrix with the eigenvalue ratios lambda_r/lambda_l */
    [B,D] = eig(Sr,Sl);
    W = transpose(B)*P;
}

Figure 4.3: Simplified code for the eeg_csp function.

This function takes as argument a cell array with the average covariance matrices of each class in each of its elements (which eeg_cov can compute and should be $C \times C$, where $C$ is the number of channels). Due to the binary nature of the CSP algorithm, only the first two elements are used. If more than two classes are considered, a one-against-all or similar approach should be taken and several calls to this function with each pair of classes should be made.

Eigenvalue decomposition is done by using the eig MATLAB built-in function.

The outputs are a $C \times C$ spatial filter matrix $W$ with a spatial filter in each row and a $C \times C$ diagonal matrix $D$ with the eigenvalues associated with each of the spatial filters in $W$. The reason
for outputting $D$ is because it is required for the filter selection described in Section 3.2.

### 4.1.4 EGG_FILTER function

The `eeg_filter` function performs bandpass filtering on trial-structured EEG data. It's simplified code is presented in Figure 4.4.

```matlab
function data = eeg_filter(data,params) {
    normBands = params.Bands/(params.SampleRate/2);
    switch(prepParams.FilterType) {
        case('iir')
            /* generates numerator 'b' and denominator 'a' coefficients
            for Chebyshev type 2 IIR filter */
            [b,a] = cheby2(prepParams.FilterOrder,20,normbands);
            for (/* k running data for each class */) {
                data(k) = filter(b,a,data(k));
                data(k) = filter(b,a,reverse(data(k)));
                data(k) = reverse(data(k));
            }
        case('fir')
            /* generates FIR filter coefficients */
            b = fir1(prepParams.FilterOrder,normbands);
            for (/* k running data for each class */) {
                data(k) = filter(b,1,data(k));
            }
        otherwise
            error('Unknown filtering type');
    }
}
```

**Figure 4.4:** Simplified code for the `eeg_filter` function.

The function receives `data`, a cell array where each element contains a 3-D array with size $T \times S \times C$, where $T$, $S$ and $C$ are the number of trials, samples and channels, respectively, and `params`, a structure with several fields which the function requires, namely `Bands`, `SampleRate`, `FilterType` and `FilterOrder`. These fields contain a two element array with the edges for the filter passband, a scalar with the frequency with which the input signal was sampled, a string with the type of filter to deploy and a scalar with it's order, respectively.

There are two types of filtering available: Finite Impulse Response (FIR) linear phase distortion and Infinite Impulse Response (IIR) zero-phase distortion. The constraints for phase distortion are related to the fact that the EEG signal is broadband (in the sense that the signal of interest is within a range rather a single frequency value) and it's temporal structure is then destroyed by non-linear phase distortion filtering. This is overcome by either filtering with a FIR filter which has linear phase distortion, thus causing a constant group delay at all frequencies and preserving time-structure, or by IIR zero-phase distortion filtering, which is obtained by filtering the signal, reversing it, running the reversed sequence again through the filter and reversing the result.

The filter is generated by using the MATLAB functions `cheby2` if the filtering type is IIR or `fir1` if it is FIR, and the data is filtered using the `filter` function. A Chebyshev type 2 filter is used because
of its flat frequency response in the passband. The filter coefficient generating functions require the passband edges to be normalized to half the sample rate, hence the use of the \texttt{normBands} variable. For the no-phase distortion filtering, the MATLAB \texttt{filtfilt} function was avoided to take advantage of vectorized expressions and thus improve performance. All of these functions are a part of the MATLAB Signal Processing Toolbox.

\texttt{Eeg.filter} outputs the input data after filtering.

### 4.1.5 \texttt{PICKFILTERS} function

The \texttt{pickfilters} functions performs spatial filter selection using the algorithms in Section 3.2. It's simplified code is presented in Figure 4.5.

```matlab
function newW = pickfilters(B,D,method,numOfSF) {
    switch(method) {
        case('traditional')
            [D, order] = sort(D);
            B = reorder(B,order);
            newW = /* take numOfSF/2 top rows and numOfSF/2 bottom rows from B */;
        case('mixed')
            [D, order] = sort(D);
            B = reorder(B,order);
            tempNewW = /* take top and bottom rows from B */;
            B = /* remove the top and bottom rows from B */;
            D = max(D,1/D);
            [D, order] = sort(D);
            B = reorder(B,order);
            newW = /* take top numOfSF-2 rows of B*/;
            newW = append(newW,tempNewW);
        case('sorted')
            D = max(D,1/D);
            [D, order] = sort(D);
            B = reorder(B,order);
            newW = /* take top numOfSF rows of B*/;
        otherwise
            error('Unknown filter selection method!');
    }
}
```

![Figure 4.5: Simplified code for the \texttt{pickfilters} function.](image)

The function receives as arguments \texttt{B}, a \( C \times C \) spatial filter matrix where \( C \) is the number of EEG channels and each row is a spatial filter, \texttt{D}, a diagonal \( C \times C \) matrix with the eigenvalues associated with each row/spatial filter of \texttt{B}, \texttt{method}, a string with the method of spatial filter selection to perform, and \texttt{numOfSF}, an integer with the number of spatial filters to select. It simply returns \texttt{newW}, a matrix with the rows from \texttt{B} which are selected by one of the algorithms.

### 4.2 Algorithm Implementations

In this section, the implementations of the algorithms described in Chapter 3 are presented. To avoid repetition, the implementation of CSP is taken as a standard and only the functions for each algorithm which differ from the corresponding CSP ones are presented.
Although a specific datamodel is not required by the BCILAB library as it only defines the order in which functions are called, the following datamodel was used and is recommended:

- Variables containing trial data or data which is extracted from the trial data should be put into cell arrays, with the data from each class in an element. This ensures that even if the dataset is asymmetrical in terms of classes (i.e., the data size is different for each class) the data can be easily stored and accessed. This notion can be extended, if necessary, to trials, by making the cell array 2-dimensional (using rows for classes and columns for trials, for example);

- All parameters are passed into functions by a single structure, which is cloned inside the crossvalidation cycle to provide storage for each crossvalidation run. At the end of each crossvalidation iteration and the simulation, the data which is relevant for the simulator to output is organized by using the `get_run_results` and `package_results` functions, respectively. Proceeding in this fashion makes the main function cleaner and reduces the time required for editing the function headers when rewriting them for an algorithm.

### 4.2.1 Common Spatial Patterns (CSP)

The CSP implementation uses the skeleton function as a basis. The structure of the skeleton function was, however, altered to include the possibility of testing several numbers of spatial filters in one simulator run and reduce the total simulation time when the number of filters varies.

#### A parse_args

The `parse_args` function serves the purpose of identifying the values with which the simulation is to be run. This function receives two inputs, `file` and `args` and generates one output, `params`, which is a structure. `file` is assumed to be a cell array where each element is a string with the name of a data file to load.

The simplified code for this function is presented in Figure 4.6.

The following parameters are available for all the algorithms:

- Number of employed spatial filters. Default is 2;
- Spatial filter selection method. Default is `traditional`;
- Crossvalidation setup. Here two options are available: either one specifies two integers \(n\) and \(k\) and the simulator generates sets for \(n\) times \(k\)-fold crossvalidation, or the indices are passed as a two element cell array. The default is to generate indices for 10 times 10-fold crossvalidation;
- Band selection method. Default is `none`.
function params = parse_args(file,args){
    params = /* setting defaults */;
    for (/* k running args two elements at a time */) {
        switch(args(k)) {
            case('NumOfSF')
                params.NumOfSF = args(k+1);
                ...
            /* cases for other parameters */
            ...
        }
    }
}

Figure 4.6: Simplified code for the parse_args function.

- Pre-processing filter band/initial band selection set. If the band selection argument is none, this parameter should be a two element array with the edges of the desired pre-processing filter passband. If not, the parameter should be a $N \times 2$ matrix where each row contains the edges for one of the bands to include in the initial band selection set. Default is $[8, 30]$ Hz.

B read_data

The read_data function reads and structures data from BCI Competition III Dataset IVa and its simplified code is presented in Figure 4.7.

function [data, params] = read_data(params){
    /* trialStarts is an array with the index of the start of each trial */
    [data,sampleRate,trialStarts] = load(params.File);
    params.SampleRate = sampleRate;
    start=0.5*params.SampleRate;
    stop=2.5*params.SampleRate;
    /* zeros(n,m,c) creates a n times m times c array of zeros */
    trials = zeros(numTrials,stop-start,numChannels);
    for (/* k running each trial */) {
        trials(k,:,:,:) = data(trialStart(k)+start:trialStart(k)+stop,:);
    }
    /* we put the data of each class in a separate element of a cell array */
    data = separate_trials_by_class(trials);
}

Figure 4.7: Simplified code for the read_data function.

Since this data is available in .mat files, the function simply loads the variables into the workspace. The EEG data read from the .mat files consists of a single $C \times T$ matrix (where $C$ is the number of channels and $T$ the total number of samples) with the data from all trials, so the function takes the start times for each trial and then restructures the data to create a $N \times S \times C$
4.2 Algorithm Implementations

3-dimensional array, where \( N \) is the number of trials and \( S \) the number of samples between defined start and stop times. These start and stop times are defined as 0.5 seconds and 2.5 seconds after the start of trial (presentation of cue). This was hard-coded due to the fact that, in this time frame, interesting activity is observable and experimenting with it is not interesting in scope of this thesis. If desired, this function can, however, be modified so that the start and stop times can be supplied as other arguments and, if omitted, default to the current values.

C  global_prep

The \( \text{global} \) \( \text{prep} \) function performs preprocessing which doesn’t require a set of training data and is thus more efficient to perform on the whole dataset at once (because there is no need to repeat it for each cross-validation set). This preprocessing consists of simple bandpass filtering which is done by calling the \( \text{eeg\_filter} \) function (see Section 4.1.4). This function’s simplified code is presented in Figure 4.8.

```matlab
function [data, params] = global_prep(data, params){
  if (/* Preprocessing is not global */) {
    return;
  }
  data = eeg_filter(data, params);
}
```

Figure 4.8: Simplified code for the \( \text{global} \) \( \text{prep} \) function.

The function arguments are \( \text{data} \), a cell array with the data of each class in an element, and \( \text{params} \), a structure with several simulator parameters.

D  local_prep

The \( \text{local} \) \( \text{prep} \) function performs preprocessing which requires training on a set of data, and consists of the procedure described in Section 3.3. This function’s simplified code is presented in Figure 4.9.

It takes the arguments \( \text{trainData} \), a cell array with training data from each class in each element, \( \text{testData} \), a cell array with the same structure as \( \text{trainData} \) but test data, and \( \text{params} \), a parameters structure where the \text{Bands} field is the initial set of bands. It’s outputs are the filtered training and testing datasets (\( \text{trainData} \) and \( \text{testData} \), respectively) and the \( \text{params} \) parameter structure.

E  get_params

The \( \text{get\_params} \) function is used to extract parameters which the BCI algorithm requires to work. For the CSP algorithm, it simply calls the \( \text{eeg\_cov} \) and \( \text{eeg\_csp} \) functions, which is illustrated by it’s simplified code in Figure 4.10.
function [trainData, testData, params] = local_prep(trainData, testData, params)
    if (/* Preprocessing is global */) {
        return;
    }

    /* generates all the nodes in the tree representation*/
    newBands = generate_band_combinations(params.Bands);

    /* Clone the parameter structure */
    tempParams = params;

    /* Maximum accuracy variable */
    maxVal = 0;
    for (/* k running params.Bands */) {
        tempData = eeg_filter(trainData, params.Bands(k));
        tempParams = get_params(tempData, tempParams);
        tempFeatures = get_features(tempData, tempParams);
        tempParams = run_classifier(tempFeatures, tempParams);
        if (tempParams.Accuracy >= maxVal) {
            best = k;
            maxVal = tempParams.Accuracy;
        }
    }

    newBands = /* get subset of of newBands which contains params.Bands(best) */
    maxVal = 0;
    for (/* k running newBands */) {
        tempData = eeg_filter(trainData, newBands(k));
        tempParams = get_params(tempData, tempParams);
        tempFeatures = get_features(tempData, tempParams);
        tempParams = run_classifier(tempFeatures, tempParams);
        if (tempParams.Accuracy >= maxVal) {
            best = k;
            maxVal = tempParams.Accuracy;
        }
    }

    params.Bands = newBands(best);
    trainData = eeg_filter(trainData, params);
    testData = eeg_filter(testData, params);
}

Figure 4.9: Simplified code for the local_prep function.

function params = get_params(data, params)
    data = eeg_cov(data);
    [params.SFMatrix, params.Eigenvalues] = eeg_csp(data);
}

Figure 4.10: Simplified code for the get_params function.
The inputs are a cell array of data (typically with data from the training set) \textit{data} and the parameter structure \textit{params} and the output is the \textit{params} structure, which is a modified version of the input argument with the same name.

\textbf{F \ get\_features}

The \textit{get\_features} function performs the feature extraction procedures for the algorithm. In the case of CSP, feature extraction is performed by applying each of the spatial filters to the EEG signal, i.e., for each spatial filter (\(C\) length array, where \(C\) is the number of channels of the original EEG signal) \(w\), a new signal \(Z\) is obtained by computing (3.2).

The simplified code for this function is presented in Figure 4.11.

```matlab
function [trainfeatures, testfeatures, params] = get_features(train,test,params,NumOfSF) {
    newSFMatrix = pickfilters(params.SFMatrix,params.Eigenvalues,params.SFSelMeth,NumOfSF);
    for (/* k running each class */) {
        /* here we turn the trial structured data into a continuous stream of EEG,
        which simplifies spatial filtering for every trial into a single matrix multiplication */
        EEGStream = concatenate(reshape(train(k)),reshape(test(k)));
        EEGStream = newSFMatrix*EEGStream;
        /* reshape into trial structure */
        EEGStream = reshape(EEGStream);
        EEGStream = variance_and_log_normalizing(EEGStream);
        [trainfeatures(k),testfeatures(k)] = separate_train_and_test_trials(EEGStream);
    }
    params.SFMatrix = newSFMatrix;
}
```

Figure 4.11: Simplified code for the \textit{get\_features} function.

The \textit{train} and \textit{test} inputs are cell arrays with EEG data from each class in a cell, while \textit{params} is a structure. The outputs are \textit{trainfeatures} and \textit{testfeatures}, two cell arrays with the feature vectors for each class in each element. The data inside each cell is structured so that each row is the feature vector for one trial. The \textit{params} output is the same as the input but with some modifications.

\textbf{G \ run\_classifier}

The \textit{run\_classifier} function handles the classifier training and testing portion of the BCI. Although the training and testing phases are typically separated in a machine learning application, since the data comes from a pre-recorded dataset and there are no real-time, continuous classification constraints, these phases were joined into a single function to ease the editing. The fact that this function handles only classification tasks also means that virtually any classifier can be used by simply rewriting the function. In addition, the standard implementation of this function accepts the use of several classifiers in a single simulator run, thus eliminating the need for several runs for each classifier. Its simplified code is presented in Figure 4.12.
function params = run_classifier(trainfeatures,testfeatures,params) {
    /* here we create a single matrix for all classes where each line
    is a feature array */
    trainFeaturesArray = join_arrays(trainfeatures);
    /* and here we produce a labels array */
    trainLabels = cell(size(trainfeatures));
    for (/* k running each element of trainfeatures */) {
        trainLabels(k) = /* produce an array with class label k for each row
        of trainfeatures */;
    }
    trainLabelsArray = join_arrays(trainLabels);
    for (/* each classifier */) {
        classifier = train_classifier(trainFeaturesArray,trainFeaturesLabel);
        for (/* k running each element of trainfeatures */) {
            trainResult = predict(classifier,trainFeatures(k));
            testResult = predict(classifier,testFeatures(k));
            for (/* m running each possible class */) {
                trainCM(m,k) = /* number of trials classified as class m */;
                testCM(m,k) = /* number of trials classified as class m */;
            }
        }
    }
    params.classification = /* data from the classification */;
}

Three classifiers are available for the default implementation:

**Näive Bayes Classifier (NBC)** Relies on estimating the data's distribution for each class and
uses this to classify later samples. This implementation uses the MATLAB Statistics Tool-
box's *NaiveBayes* object;

**Linear Discriminant Analysis (LDA)** Projects the features into a single dimension space and
classifies them according to their position in this new space. It is implemented using the
MATLAB Statistics Toolbox’s *classify* function with the discriminant type set to linear;

**Support Vector Machine (SVM)** Computes a hyperplane in the feature space which is under-
stood to maximize the gap between features of different classes. Later samples are then
classified by their location in this space in relation to the previously computed hyperplane.
For this default implementation, the MATLAB Bioinformatics Toolbox’s functions *svmtrain*
and *svmclassify* are used.

The function takes as inputs *trainfeatures* and *testfeatures*, which are two cell arrays with
features for each class in each element. The features in each cell are in a matrix where each row
contains the feature vector for a trial. The third input argument is the parameter structure *params*.
The output is the *params* structure with the classification results (a confusion matrix) added to it.
4.2 Algorithm Implementations

**H get_run_results**

The *get_run_results* function doesn’t perform any actual computations and is only used to organize the results from each crossvalidation run of the simulator, as usually the same parameter structure is passed into every function and modified inside it. It was thus found easier to create a single function with the purpose of “cleaning up” other functions’ output, allowing the developer to focus on the algorithm and managing metadata in separate phases. This decision was made because, although it makes the parameter structure larger (because unused parameters are kept), this increase is negligible when considering the huge size of numerical data (for example, a single second of EEG from 10 channels at a sampling rate of 100 Hz with double precision takes about 8 kB, while a 50 character string is 50 bytes).

For CSP, the results saved for each run are the spatial filter matrix, the pre-processing filter’s passband edge frequencies, the time spent on the run and the results of classification for that run, which are a structure containing the confusion matrices for the training and testing sets for each of the selected classifiers.

**I package_results**

The *package_results* function operates similarly to *get_run_results* but it runs over the data from all crossvalidation runs and it performs some computation to obtain some basic statistics on the simulation results. These basic statistics are generated for each (classifier, number of spatial filters) pair. The first of these statistics is the mean accuracy over all crossvalidation sets, computed as

\[
\bar{Acc} = \frac{\sum_{n=1}^{N} \text{tr}(C_n)}{\sum_{n=1}^{N} \text{sum}(C_n)},
\]

where \(C_n\) is the confusion matrix of trial \(n\) and the \(\text{sum}\) operator sums all the elements of the matrix it is applied to. This statistic is computed for running the trained classifier on the training set and on the testing set.

The other basic statistics are the minimum and maximum accuracies, computed as

\[
\begin{align*}
\text{Acc}_{\text{min}} &= \min \left( \frac{\text{tr}(C_n)}{\text{sum}(C_n)} \right), \quad n = 1, 2, \ldots, N, \\
\text{Acc}_{\text{max}} &= \max \left( \frac{\text{tr}(C_n)}{\text{sum}(C_n)} \right), \quad n = 1, 2, \ldots, N.
\end{align*}
\]

These are computed for both the training and testing sets as well.

The remaining parameters which are outputted are the file(s) from which the data was read, the pre-processing method, the pre-processing filter type and order, the number of iterations and
4. **BCILAB — Brain-Computer Interface (BCI) MATLAB Library**

the number of crossvalidation sets for each of the iterations, the spatial filter selection method and the number of spatial filters to test, the indices used for cross-validation (these are saved mainly for ensuring fairness when running different algorithms) and the results for each run (included in case additional statistics need to be computed).

### 4.2.2 Common Spatio-Spectral Patterns (CSSP)

**A parse_args**

The CSSP algorithm has an additional parameter to control the number of samples delayed in the signal copy which is appended to the original data. The default delay is 15 samples.

**B read_data**

The `read_data` function was modified for the CSSP implementation such that the simplified code is now the one presented in Figure 4.13.

```matlab
function [data, params] = read_data(params) {
    /* trialStarts is an array with the index of the start of each trial */
    [data, sampleRate, trialStarts] = load(params.File);
    params.SampleRate = sampleRate;
    start = 0.5*params.SampleRate;
    stop = 2.5*params.SampleRate;
    trials = alloc(numTrials, stop-start, 2*numChannels);
    for (/* k running each trial */) {
        trials(k,:,1:numChannels) = data(trialStart(k)+start:trialStart(k)+stop,:);
        trials(k,:,numChannels+1:2*numChannels) = data(trialStart(k)+start+params.Delay:trialStart(k)+stop+params.Delay);
    }
    /* we put the data of each class in a separate element of a cell array */
    data = separate_trials_by_class(trials);
}
```

Figure 4.13: Simplified code for the `read_data` function of the CSSP implementation.

In this implementation, the data matrix is enlarged to include a delayed copy of the signal as a new set of EEG channels.

**C package_results**

For the result packaging, this implementation includes the number of delayed samples as a parameter.
4.2 Algorithm Implementations

4.2.3 Common Sparse Spectral Spatial Patterns (CSSSP)

A  parse_args

For the CSSSP algorithm, an additional argument can be supplied, which is the length of the FIR filter to be optimized by the algorithm. The default value is 16.

B  local_prep

The local_prep function is not included for the implementation of CSSSP because of the algorithm’s already very computational heaviness.

C  get_params

The simplified code for the CSSSP version of get_params is provided in Figure 4.14.

```matlab
function [train, test, params] = get_params(train, test, params){
    params.b = zeros(auxParams.FilterSize,1);
    params.b(1) = 1;
    /* @costFun represents the cost function */
    params.b = fminsearch(@costFun,auxParams.b);
    for /* k running the elements of train */) {
        train(k) = filter(params.b,1,train(k));
        test(k) = filter(params.b,1,test(k));
    }
    trainCov = eeg_cov(train);
    [params.SFMatrix,params.Eigenvalues] = eeg_csp(trainCov);
}
```

Figure 4.14: Simplified code for the get_params function in the CSSSP implementation.

The cost function is not the one presented in (3.28) but a functionally equivalent which consists of computing the sum of the features obtained through the CSP algorithm after the input data has been filtered by a FIR filter (which is the optimization variable). This cost function is supplied to the MATLAB Optimization Toolbox function fminsearch, which uses the simplex search method.

Of particular notice is the inclusion of the training and testing data as input and output parameters. This was done so that the data is filtered with the filter obtained from the optimization prior to calling the get_features function, thus avoiding rewriting that function as well.

D  get_run_results

An additional parameter is saved for each crossvalidation run, the spectral filter obtained through numerical optimization.
For the CSSSP implementation, the spectral filter’s length is added as a field to the output structure.

### 4.2.4 Combined Frequency Band Spatial Patterns (CFBSP)

#### A parse_args

Since CFBSP assumes that frequency selection is always performed, the `parse_args` function is modified so that the default initial bands are the 4 Hz wide bands centered at \{6, 8, 10, 12, 14, 16, 18, 20, 22\}.

#### B local_prep

The simplified code of the implementation of `local_prep` for CFBSP is presented in Figure 4.15. This version selects the two bands of the initial set with best performance, then filters the data into each of those bands and proceeds to join in them into a single dataset. A noteworthy difference between this implementation and the one in other algorithms is that there is only one stage of band selection.

### 4.2.5 Spatial Patterns for Maximal Separability (SPMS)

#### A get_params

The implementation of `get_params` is done to account for the solving of the numerical optimization problem which is at the core of the method. One particular notice must be made that, because the eigendecomposition is not used, the optimization must be run every time the desired number of spatial filters changes (i.e., there is no guarantee that selecting 3 filters will yield the set of filters obtained by selecting 2 filters plus one other filter). Its simplified code is presented in Figure 4.16. The `get_params` function relies on the `costcalc` function to evaluate the value of the cost function, supplying it to the Optimization Toolbox `fminsearch` function which solves the optimization problem using the simplex method. The `costcalc` function applies the cost function described in (3.39).

### 4.2.6 get_features

For SPMS, since only a single number of filters can be selected for each simulator run, the `get_features` function is modified to remove the step where filters are picked from the spatial filter matrix.
4.2 Algorithm Implementations

```matlab
function [trainData, testData, params] = local_prep(trainData, testData, params) {
    if (/* Preprocessing is global */) {
        return;
    }
    /* generates all the nodes in the tree representation*/
    newBands = generate_band_combinations(params.Bands);
    /* Clone the parameter structure */
    tempParams = params;
    /* Maximum accuracy variables */
    maxVal1 = 0;
    maxVal2 = 0;
    best1 = 0;
    for (/* k running params.Bands */) {
        tempData = eeg_filter(trainData, params.Bands(k));
        tempParams = get_params(tempData, tempParams);
        tempFeatures = get_features(tempData, tempParams);
        tempParams = run_classifier(tempFeatures, tempParams);
        if (tempParams.Accuracy >= maxVal1) {
            best2 = best1;
            maxVal2 = maxVal1;
            best = k;
            maxVal1 = tempParams.Accuracy;
        } else if (tempParams.Acurracy >= maxVal2) {
            best2 = k;
            maxVal2 = tempParams.Accuracy;
        }
    } [params1.params2] = params;
    params1.Bands = params.Bands(best,:);
    params2.Bands = params.Bands(best2,:);
    trainData1 = eeg_filter(trainData, params1);
    testData1 = eeg_filter(testData, params1);
    trainData2 = eeg_filter(trainData, params2);
    testData2 = eeg_filter(testData, params2);
    params.Bands = join(params1.Bands, params2.Bands);
    trainData = join(trainData1,trainData2);
    testData = join(testData1,testData2);
}
```

Figure 4.15: Simplified code for the `local_prep` function implementation for the CFBSP algorithm.

```matlab
function params = get_params(data, params){
    covMat = eeg_cov(train);
    [W,D] = eeg_csp(covMat);
    W = pickfilters(W,D,'traditional',params.NumOfSF);
    /* @costcalc is the cost function */
    W = fminsearch(@costcalc,W,options);
    params.SFMatrix = W;
}
```

Figure 4.16: Simplified code for the SPMS implementation of the `get_params` function.
4. BCILAB — Brain-Computer Interface (BCI) MATLAB Library

4.2.7 Spatial Filter Residual Analysis (SFRA)

A get_features

The implementation of get_features for SFRA adds the training of AR models to the standard implementation, as well as the filtering of the data with those models.

The simplified code for this function is presented in Figure 4.17.

```matlab
function [trainfeatures, testfeatures, params] = get_features(train, test, params, NumOfSF) {
    /* D is an array with the eigenvalue associated with each spatial filter */
    [newSFMatrix, D] = pickfilters(params.SFMatrix, params.Eigenvalues, params.SFSelMeth, NumOfSF);
    ARModels = alloc(params.Order+1, length(D));
    for /* k running each spatial filter */) {
        /* we train the model using the data from the class for which the variance is minimized. */
        if /* Eigenvalue is < 1 */) {
            /* class 1 trials' variances are minimized */
            ARModels(:, k) = arburg(newSFMatrix(k, :) * mean(train(1)), ORDER);
        } else {
            /* class 2 trials' variances are minimized */
            ARModels(:, k) = arburg(newSFMatrix(k, :) * mean(train(2)), ORDER);
        }
    }

    for /* k running each class */) {
        /* here we turn the trial structured data into a continuous stream of EEG, which simplifies spatial filtering for every trial into a single matrix multiplication */
        EEGStream = concatenate(reshape(train(k)), reshape(test(k)));
        EEGStream = newSFMatrix * EEGStream;
        /* reshape into trial structure */
        EEGStream = reshape(EEGStream);
        for /* m running each AR model */) {
            EEGStream(:, :, m) = filter(ARModels(:, m), EEGStream(:, :, m));
        }
        EEGStream = variance_and_log_normalizing(EEGStream);
        [trainfeatures(k), testfeatures(k)] = separate_train_and_test_trials(EEGStream);
    }
    params.SFMatrix = newSFMatrix;
}
```

Figure 4.17: Simplified code for the SFRA implementation of the get_features function.

The function uses the eigenvalues outputted by pickfilters to select the data to be used in the AR model training. The selected data is spatially filtered and averaged to generate a single trial per spatial filter, which is in turn used as input to the MATLAB Signal Processing Toolbox function arburg, which computes the AR model for that trial.

For feature calculation, the generated AR models are used for temporal filtering of the signals resulting from spatial filtering. Again, each AR model is used to filter only the result corresponding to the same spatial filter used in the training procedure.
4.2 Algorithm Implementations

Summary

In this chapter, BCILAB, a collection of developed MATLAB functions was presented. This library includes a Brain-Computer Interface (BCI) skeleton function, which defines the structure of a BCI simulator and can be used to easily and quickly implement new algorithms. The skeleton function breaks a BCI into functionally separated steps and runs on even the most basic MATLAB installations for maximum compatibility. To aid development, some implementations of commonly used operations in the scope of CSP-BCIs, such as filtering and computation of covariance matrices, are made available. Also, implementations of the algorithms described in Chapter 3 were developed and are used to obtain the experimental results presented in the next section.
4. *BCILAB* — Brain-Computer Interface (BCI) MATLAB Library
5

Experimental results

Contents

5.1 Testing framework ........................................... 56
5.2 Algorithm results ........................................... 57
5.3 Effects of Frequency Band Selection ...................... 78
5.4 Comparison of classifiers ................................. 80
5.5 Comparison of BCI algorithms ............................ 80
5. Experimental results

In this chapter, a description of the experimental procedures and their results is presented. This is broken into three sections, with the first describing the experimental setup used for running the simulations, that is, the dataset as well as the algorithms and simulation parameters used for the tests. The second section analyses each algorithm separately, presenting outputs and confronting them with the descriptions in 3, and the third studies the effects of the proposed frequency band selection procedure on those algorithms. The fourth section presents a comparison between classifiers, while the fifth and last section presents a comparison between the tested algorithms.

5.1 Testing framework

All of the experimental work presented was done on the BCI Competition III dataset IVa [23]. This dataset comprises MI trial data from five healthy subjects with two classes, right hand and right foot. It was chosen because the featured EEG data is recorded into 118 channels at a rate of 100 Hz and thus provides two characteristics, the first being that CSP-based methods can take advantage of the large number of electrodes and the second that algorithm efficiency becomes a concern because of the large amount of data (a total of $118 \times 100 = 11800$ samples/s).

The data used from each trial corresponds to the time frame $[0.5 - 2.5]$ s after the cue (yielding a total of $11800 \times 2 = 23600$ samples per trial) and, though all three possible classifiers (NBC, LDA and SVM) were employed, unless noted, the results for the NBC are presented. This classifier was chosen because, in the case of the SVM MATLAB functions, an “all or nothing” approach with regards to solving the underlying hyperplane optimization problem is taken: if the limit on the number of iterations is reached (which is a mandatory parameter and cannot be set to be infinity), the function exits with an error. Although this situation is handled in the simulator so that the whole simulation isn’t interrupted, it made result gathering for the algorithm very unreliable. In the case of LDA, an error is generated, on some occasions, while trying to obtained a pooled variance estimate for the features. No workaround for this could be developed, as the assumption that the variance estimates can be pooled together is not guaranteed to be true.

To validate the simulation results, 10-times 10-fold cross-validation was performed and the same partitions were used on all algorithms.

The following algorithms were tested:

- CSP;
- CSSP;
- CSSSP;
- CFBSP;
- SPMS;
5.2 Algorithm results

This section presents the individual simulation results for each of the CSP-based algorithms presented in Chapter 3. For each algorithm, the spatial filter selection methods presented in Section 3.2 are tested to provide deeper insight into the results. The exception to this is the SFRA algorithm since it directly computes the desired number of spatial filters.

5.2.1 Common Spatial Patterns (CSP)

Several simulations of the CSP algorithm were run with varying parameters, namely spatial filter selection techniques, number of selected spatial filters and whether to perform frequency band tuning. For all three spatial filter selection methods (traditional, mixed and sorted), the 6 lowest possible number of spatial filters were tested, that is, for the traditional method, the number of spatial filters varied in \( \{2, 4, 6, 8, 10, 12\} \), while, for both mixed and sorted, it varied in \( \{2, 3, 4, 5, 6, 7\} \).

To provide some deeper insight into how CSP works, two spatial filters considered representative in the discrimination of the two MIs are presented in Figure 5.1 by mapping each filter coefficient with the position over the scalp of the respective electrode used for acquisition.

![Spatial Filter for the right hand MI.](image1)

![Spatial Filter for the right foot MI.](image2)

**Figure 5.1:** Spatial Filters for each MI obtained through the CSP method. The bars on the right-hand side of each filter map the colours to the numerical weights given to channels.

Analysis of the two filters shows us that the obtained filters are supported by the physiological phenomena described in Chapter 2, because coefficients tend to be higher in absolute value for electrodes positioned over the areas of the brain associated with the MI being performed. More specifically, the brighter and darker portions of the filters are concentrated over the center-left
5. Experimental results

portion of the brain for the right hand MI (Figure 5.1(a)) and, for the right foot MI (Figure 5.1(b)), despite the pattern being more scattered, greater emphasis is given to activity over the farther left portion of the scalp.

The results for the simulation of CSP using the traditional spatial filter method are shown as a bar graph in Figure 5.2.

![Bar graph showing accuracy on training and testing sets](image)

(a) Accuracy on the training set. (b) Accuracy on the testing set.

Figure 5.2: BCI classification results for the CSP algorithm, using the traditional spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

These results allow us to make a number of observations, such as the fact that accuracy drops significantly when switching from the training data set to the testing one. This hints that the CSP algorithm overfits the training data, i.e., it fails to identify the features of the data which truly distinguish the classes and thus includes in it’s model phenomena which are specific to the training data only and do not describe the testing data. Another hint for this is that increasing the number of filters translates into a steady increase in accuracy on the training set, whereas adding filters over the testing set only improves performance up to a certain point, at which accuracy starts decreasing.

Similar conclusions can be drawn when the mixed spatial filter selection is used, with the results being presented in Figure 5.3. For this method, however, there is a noticeable difference in how accuracy relates to an increase in the number of employed spatial filters, as the accuracy peaks tend to be smoother. This is most likely related to the finer grained selection range for the number of filters, since the mixed method allows us to add a single filter at a time while the traditional requires a minimum increase of 2 filters. Also interesting is the fact that, although the mixed algorithm’s accuracies on the testing data are comparable with the traditional algorithm’s, they are lower on the training set. This suggests that mixed performs in a more equal manner on both the training and testing sets (which can be understood as diminished overfitting).

Finally, the results for the sorted method are displayed in Figure 5.4. These results are very alike to the ones for the mixed method, so the same conclusions apply for both methods.
5.2 Algorithm results

Figure 5.3: BCI classification results for the CSP algorithm, using the mixed spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

Figure 5.4: BCI classification results for the CSP algorithm, using the sorted spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.
5. Experimental results

Using the same data employed in Figures 5.2, 5.3 and 5.4, a comparison of the spatial filter selection techniques can be made by plotting the accuracy results for every subject and every number of spatial filters, from one algorithm against another algorithm. To ensure fairness, the comparison is only made when both algorithms use the same number of spatial filters and the subject is the same. This is shown in Figure 5.5. These graphs for the testing sets allow us to conclude that the mixed method provides better results for the same number of filters, as it always produces accuracies which are at least comparable to any of the other algorithms. In particular, it is inferior to the traditional algorithm in only two instances, and even then the difference is around 1%, while it can outperform the same algorithm by a difference of over 5%. Against the sorted algorithm, the mixed method almost always performs better and, again, when performing worse, the difference is only noticeable in one case, by 4%. Of notice is also the fact that the mixed method appears to be inferior when only the training set’s results are considered, both against traditional and sorted, proving that this method is also more robust to overfitting. Although the sorted method does not perform as well as the mixed one, it does, on occasion, produce very positive improvements over traditional.
5.2 Algorithm results

Figure 5.5: Comparison of filter selection methods with no frequency band tuning applied, for the CSP algorithm, every subject and every number of spatial filters. Above the line means the algorithm on the vertical axis is better.
5. Experimental results

5.2.2 Common Spatio-Spectral Patterns (CSSP)

For the CSSP algorithm, simulations were run while varying the time delays of the appended signals between 1 and 15 samples, which, at a sampling rate of 100 Hz, equal delays in the interval $[10, 150]$ ms. This was done in addition to the same variation in number of spatial filters used for CSP simulations. In order to outline how CSSP works, the evolution of BCI accuracy with the increasing time delays is shown in Figure 5.6. Although some subjects display a more noticeable trend than others, there appears to be an oscillation in accuracies with varying frequency for each subject. This is in agreement with the premise of CSSP, as certain time delays can enhance the oscillatory characteristics of the EEG signal. The fact that the frequency is subject specific further justifies this view, especially if one considers that accuracies peak at a time delay of around $100$ ms $= 10$ Hz, which falls inside the Alpha band typically associated with MI.

Additionally, the three spatial filter selection methods were tried, and the comparative results are presented in Figure 5.7. In the CSSP context, the \textit{sorted} algorithm is clearly inferior, while the choice between the \textit{mixed} and \textit{traditional} methods seems to be dependent on the situation. More specifically, this could imply that the filter selection techniques should be subject oriented, as each algorithm is superior in a well defined area. Another visible trend is that the \textit{traditional} method, clearly superior on the training set, always loses some advantage in comparison to the other algorithms on the testing set, revealing overfitting issues.
5.2 Algorithm results

Figure 5.6: Variation in the accuracy of the CSSP algorithm when different time delays are used.

(a) Traditional spatial filter selection with 2 filters, training set.

(b) Traditional spatial filter selection with 2 filters, testing set.
5. Experimental results

Figure 5.7: Comparison of filter selection methods with no frequency band tuning applied, for the CSSP method, every subject and every number of spatial filters. Above the line means the algorithm on the vertical axis is better.
5.2 Algorithm results

5.2.3 Common Sparse Spectral Spatial Patterns (CSSSP)

The CSSSP algorithm was tested with the same spatial filter parameters as CSP and the length of the FIR filter obtained from optimization was set to 16 coefficients. Example parameters computed by CSSSP are shown in Figure 5.8. These spatial filters are very similar to the ones obtained through CSP (see Figure 5.1), although the filter for right foot MI is not as disperse in the former case. The spectral filter, which is the fundamental difference between CSP and CSSSP, makes the frequency regions around 8 and 30 Hz stand out and is in agreement with the physiological knowledge, since it mostly emphasizes portions of the Alpha and Beta frequency ranges.

The accuracy results as a function of the number of spatial filters are presented, for each spatial filter selection method, in Figures 5.9, 5.10 and 5.11. These results suggest that CSSSP nearly eliminates overfitting, as the accuracy on the testing set is very close to the accuracy on the training set. Possibly as a result of this, using a larger number of spatial filters always translates into better accuracy, even for the largest tested number of filters and on the testing set. As such, some more insightful conclusions could have been drawn had larger number of filters been tested. This wasn’t done because of the very long simulation times when this algorithm is used.

Also, the comparison of filter selection methods is shown in Figure 5.12. In this situation, although the results are close for traditional and mixed, the traditional method is always superior, both on the training and testing sets. Again, and even though it sometimes outperforms mixed, the sorted algorithm appears to be the worst possible choice because it never performs in a clearly superior way and it is significantly inferior in many situations.
5. Experimental results

Figure 5.8: Spatial Filters for each MI obtained through the CSP method. Spatial Filters for each MI and Spectral Filter obtained through the CSSSP method. The bars on the right-hand side of each spatial filter map the colours to the numerical weights given to channels.
5.2 Algorithm results

![Bar charts showing classification accuracy for the CSSSP algorithm using different spatial filter selection methods.](image)

**Figure 5.9**: BCI classification results for the CSSSP algorithm, using the *traditional* spatial filter selection method. Each bar corresponds to a number of employed spatial filters.

![Bar charts showing classification accuracy for the CSSSP algorithm using different spatial filter selection methods.](image)

**Figure 5.10**: BCI classification results for the CSSSP algorithm, using the *mixed* spatial filter selection method. Each bar corresponds to a number of employed spatial filters.

![Bar charts showing classification accuracy for the CSSSP algorithm using different spatial filter selection methods.](image)

**Figure 5.11**: BCI classification results for the CSSSP algorithm, using the *sorted* spatial filter selection method. Each bar corresponds to a number of employed spatial filters.
Figure 5.12: Comparison of filter selection methods with no frequency band tuning applied, for the CSSSP algorithm, every subject and every number of spatial filters. Above the line means the algorithm on the vertical axis is better.
5.2.4 Combined Frequency Band Spatial Patterns (CFBSP)

Again, for the CFBSP algorithm, the number of spatial filters was varied in the same way as for CSP. Since this algorithm relies on a frequency tuning scheme, it is necessary to specify the parameters used to generate the initial band set. For this algorithm, the $[4 - 24]$ Hz frequency range was used with a 4 Hz minimum bandwidth and a 2 Hz granularity.

Samples of the spatial filters, as well as the chosen frequency bands obtained for one run of the simulator are presented in Figure 5.13.

![Spatial filters](image)

Figure 5.13: Spatial filters obtained using CFBSP for each of the frequency bands and MIs. Corresponding frequency bands are indicated under each filter and the bars on the right-hand side map the colours to the numerical weights given to channels.

The shown filters are very similar to filters obtained with previous methods and thus the same conclusions apply. The premise of CFBSP is also shown to be true because, despite the filters for right hand MI being almost identical (emphasizing the same area of the scalp) in both frequency bands (Figures 5.13(a) and 5.13(c)), there is a noticeable difference for the filters for right foot MI (Figures 5.13(b) and 5.13(d)). The chosen frequency bands are also plausible, focusing mostly in
the higher and lower ends of the Alpha and Beta frequency bands, respectively.

The results according to number of spatial filters are presented in Figures 5.14, 5.15 and 5.16, for the traditional, mixed and sorted filter selection methods, respectively.

Figure 5.14: BCI classification results for the CFBSP algorithm, using the traditional spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

Figure 5.15: BCI classification results for the CFBSP algorithm, using the mixed spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

The results feature very impressive accuracies when the algorithm is applied on the training set, however, these do not carry over to the testing set, thus revealing a severe overfitting problem. Despite this problem, accuracy over the testing set is reasonable. Another noteworthy point is the increase in performance when the number of spatial filters increases, for the mixed and sorted approaches to filter selection, which is not seen on the traditional method results. This increase is visible up to the maximum number of filters tested and suggests that larger numbers of spatial filters should be tested for these two methods. Also, the sorted method applied to the data of subject al resulted in abnormally low accuracy.
Figure 5.16: BCI classification results for the CFBSP algorithm, using the \textit{sorted} spatial filter selection method and \textbf{no} frequency tuning. Each bar corresponds to a number of employed spatial filters.

Plotting the results of the methods against each other yields Figure 5.17. The plots lead us to conclude that, for CFBSP, the \textit{mixed} method is the superior filter selection algorithm on the testing set, with \textit{traditional} being the second best and the \textit{sorted} being the worst. For the training set, the \textit{traditional} method again outperforms the other methods, which implies, as in most other cases, that it overfits more, as the trend is for the other methods to narrow the accuracy gap on the testing set.
5. Experimental results

Figure 5.17: Comparison of filter selection methods for the CFBSP method, every subject and every number of spatial filters. Above the line means the algorithm on the vertical axis is better.
5.2 Algorithm results

5.2.5 Spatial Filter Residual Analysis (SFRA)

Simulations of the SFRA algorithm followed the same standards in terms of filter selection as in the previous sections. The prediction model order was set to 30.

This algorithm produces both spatial filters and AR models (which can be interpreted as spectral filters), shown in Figures 5.18 and 5.19.

![Spatial filter for the right hand MI](image1)

(a) Spatial filter for the right hand MI

![Spatial filter for the right foot MI](image2)

(b) Spatial filter for the right foot MI

Figure 5.18: Spatial filters obtained using the SFRA algorithm, for each of the MIs. The bars on the right-hand side of each spatial filter map the colours to the numerical weights given to channels.

Since the algorithm computes the spatial filters exactly as the CSP algorithm, the filters that are outputted are identical, for equal datasets. Thus, the shown spatial filters are also physiologically plausible, emphasizing the areas of the scalp associated with each of the MIs (the center left area for right hand and the farther left area, although the pattern is more disperse, for right foot). The AR models, despite being very similar, differ the most around the 8 Hz frequency. This hints that it is around this frequency that trials of different classes differ, which is believable as it fits in the Alpha frequency band, the most important in terms of MI.

Figures 5.20, 5.21 and 5.22 show the results for each of the spatial filter selection methods.

The results for SFRA show the same trends as most other algorithms. Although the tested numbers of spatial filters are enough to find a peak in the traditional method's accuracy, in some cases, both mixed and sorted keep improving with more spatial filters and thus brings us to the conclusion that larger numbers of filters might further increase performance.

Comparing each filter selection method pairwise results in the plots presented in Figure 5.23.

The plots for the testing set indicate that neither algorithm is superior, though sorted is clearly inferior in one instance. On the training set the conclusion is mostly the same and it is noteworthy that the traditional method only performs slightly better than mixed, contrary to its tendency to be significantly better on the training set and comparable on the testing set.
5. Experimental results

Figure 5.19: Predictors obtained using the SFRA algorithm, for each of the MI's.
5.2 Algorithm results

Figure 5.20: BCI classification results for the SFRA algorithm, using the traditional spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

Figure 5.21: BCI classification results for the SFRA algorithm, using the mixed spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.

Figure 5.22: BCI classification results for the SFRA algorithm, using the sorted spatial filter selection method and no frequency tuning. Each bar corresponds to a number of employed spatial filters.
5. Experimental results

Figure 5.23: Comparison of filter selection methods with no frequency band tuning applied, for the SFRA method, every subject and every number of spatial filters. Above the line means the algorithm on the vertical axis is better.
5.2 Algorithm results

5.2.6 Spatial Patterns for Maximal Separability (SPMS)

Due to the different nature of the SPMS algorithm, a single simulation was run with 2 spatial filters. This is because a full SPMS simulation has to be run for each number of spatial filters and each simulation of the algorithm is very lengthy. Examples of spatial filters obtained using this method are presented in Figure 5.24.

![Spatial Filters](image)

Figure 5.24: Spatial Filters for each MI obtained through the SPMS method. The bars on the right-hand side of each filter map the colours to the numerical weights given to channels.

The filters are very similar to the ones computed through CSP, although they seem to be smoother (the filter in 5.24(b) doesn’t seem as dispersed, for example), and so the conclusions apply as the ones for CSP.

Figure 5.25 presents the accuracy results for the SPMS algorithm.

![Accuracy Results](image)

Figure 5.25: BCI classification results for the SPMS algorithm. Each bar corresponds to a number of employed spatial filters.

The results are again similar to the CSP ones, with accuracy dropping from the training set to the testing set (a sign of overfitting).
5. Experimental results

5.3 Effects of Frequency Band Selection

In this section, the results for algorithms using the band selection method of Section 3.3 are presented and compared to the results using the default pre-processing setup. Because not all algorithms were implemented with the possibility of using band selection, only the following are considered:

- CSP;
- CSSP;
- SFRA.

The used spatial filter selection method is mixed, since the results in Section 5.2 show it to suffer less from overfitting while providing comparable results to traditional and thus be better to assess the impact of the frequency band selection. The number of spatial filters is then varied in the integer values within the \([2 - 7]\) interval. To distinguish between algorithms, when band selection is performed, the algorithm’s name will be prefixed with FT (for example, the frequency tuned version of CSP will change to FTCSP).

The average accuracy across all crossvalidation runs is used as a performance metric. Comparison plots between regular and frequency tuned algorithms are presented in Figures 5.26, 5.27 and 5.28 for CSP, CSSP and SFRA, respectively.

(a) FTCSP vs CSP, training set.  
(b) FTCSP vs CSP, testing set.

Figure 5.26: Comparison of CSP and FTCSP. Above the line means the algorithm on the vertical axis is better.

The plots show that frequency tuning can be used to improve the performance of these algorithms. Despite the fact that they don’t consistently outperform the original, the algorithms with frequency tuned pre-processing can show very significant increases in accuracy. An important observation is that tuned pre-processing performs very well on the training set (which had already been clear when analysing the results of CFBSP) and then drops when switching to the testing...
5.3 Effects of Frequency Band Selection

Figure 5.27: Comparison of CSSP and FTCSSP. Above the line means the algorithm on the vertical axis is better.

Figure 5.28: Comparison of SFRA and FTSFRA. Above the line means the algorithm on the vertical axis is better.
5. Experimental results

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number of delayed samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>10</td>
</tr>
<tr>
<td>al</td>
<td>12</td>
</tr>
<tr>
<td>av</td>
<td>9</td>
</tr>
<tr>
<td>aw</td>
<td>12</td>
</tr>
<tr>
<td>ay</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5.1: Number of delayed samples in the CSSP algorithm for each subject.

set, which leads to the conclusion that it is overfitting the training data. This particular fact is most likely a consequence of the very simple procedure used for frequency selection and could be improved by a crossvalidation procedure, although this option would lead to increased computational effort.

5.4 Comparison of classifiers

The simulator developed in the scope of this thesis allows the use of three classifiers, NBC, LDA and SVM. However, as stated in Section 5.1, running the LDA and SVM classifiers' MATLAB functions sometimes produces errors and, as such, the only algorithm with data allowing the comparison of classifiers is CSP.

Figure 5.29 shows the comparison plots between these three classifiers. The plots show that LDA achieves the best results, while NBC and SVM are seemingly tied as second best. Moreover, SVM, which is clearly superior to the other classifiers on the training set, drops significantly in accuracy on the testing set, leading to the conclusion that it is more affected by overfitting issues. These results are not particularly surprising, since the way in which CSP optimizes the spatial filters is very well suited to the characteristics of LDA and SVM. It would have been expected, however, that SVM would perform closer to LDA on the testing set, as both are somewhat related, but this could be a result of the training set being too small for the SVM classifier rather than a proof of it's inadequacy.

5.5 Comparison of BCI algorithms

In this section, a comparison of the BCI algorithms presented in this thesis is made.

For this, CSP is taken as the baseline algorithm and all other algorithms are compared individually against it. The frequency tuning procedure is not performed, since it has been addressed in the previous section, and, where possible, the spatial filter selection algorithm is mixed. For the CSSP algorithm, which also requires a delay parameter, the parameter was chosen individually for each subject based on the best delays using 2 spatial filters. The delay associated with each subject is presented in Table 5.1.

The comparison plots are shown in Figures 5.30 to 5.33. From the figures, we see that all the
5.5 Comparison of BCI algorithms

Figure 5.29: Comparison of classifiers for the CSP algorithm, every subject and every number of spatial filters using the mixed filter selection method. Above the line means the algorithm on the vertical axis is better.
5. Experimental results

Figure 5.30: Comparison of CSP and CSSP. Above the line means the algorithm on the vertical axis is better.

Figure 5.31: Comparison of CSP and CSSSP. Above the line means the algorithm on the vertical axis is better.
5.5 Comparison of BCI algorithms

Figure 5.32: Comparison of CSP and CFBSP. Above the line means the algorithm on the vertical axis is better.

(a) CFBSP vs CSP, training set.  
(b) CFBSP vs CSP, testing set.

Figure 5.33: Comparison of CSP and SFRA. Above the line means the algorithm on the vertical axis is better.

(a) SFRA vs CSP, training set.  
(b) SFRA vs CSP, testing set.
5. Experimental results

![Figure 5.34: Comparison of CSP and SPMS. Above the line means the algorithm on the vertical axis is better.](image)

Figure 5.34: Comparison of CSP and SPMS. Above the line means the algorithm on the vertical axis is better.

algorithms can provide comparable performance to the CSP algorithm. The CSSP algorithm only performs worse than CSP in a few instances and even then is is very close, while it is superior in almost every case, sometimes with noticeable improvements. CSSSP performs very well, as it produces great improvements and is never inferior to CSP, particularly on the testing set. The results on the training set for CFBSP are the best among all algorithms (none of the subjects shows under 85% accuracy) but these drop a lot when switching to the testing set, reflecting the severe overfitting that occurs due to the very simple frequency selection method. Despite this, CFBSP still achieves good results, particularly from a worst case point of view since accuracy is never under 65%. SFRA performs slightly below CSP on most occasions, although it can be capable of outperforming it on many occasions. Surprisingly, adding the AR models on the training set actually leads to lower accuracy rates.

To provide a unified comparison of all the algorithms, the results using the parameters which maximize the mean crossvalidation accuracy for each algorithm are presented in the bar graphs of Figure 5.35. The simulation parameters which yielded these results are shown in Table 5.2.

These graphs confirm the conclusions drawn above, with CSSSP producing the best performance overall. Notice that all subjects achieve over 80% accuracy and 4 out of the 5 achieve over 95%. As for the other algorithms, with the exception of SPMS, results on the training set are very close. On the testing set, results vary a bit and no definite conclusion can be taken on which algorithm is superior (excluding CSSSP). A notice should be made here that the comparison made here is unfair to SPMS since no parameters were varied for its simulation.

To more fairly assess the behaviour of SPMS, Figure 5.36 is presented, where all algorithms are restricted to using 2 spatial filters.

Analysing these results show us that SPMS is actually on par with the remaining algorithms and that simulating with additional spatial filters can yield even better results. A fairly important
5.5 Comparison of BCI algorithms

Figure 5.35: Comparison of the maximum obtained accuracy for several BCI algorithms.

Figure 5.36: Comparison of the obtained accuracy for several BCI algorithms when the number of spatial filters is restricted to 2.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Subject</th>
<th>Number of spatial filters</th>
<th>Spatial filter selection method</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSP</td>
<td>aa</td>
<td>4</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSP</td>
<td>al</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSP</td>
<td>av</td>
<td>6</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSP</td>
<td>aw</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSP</td>
<td>ay</td>
<td>3</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSSP</td>
<td>aa</td>
<td>4</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSSP</td>
<td>al</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSSP</td>
<td>av</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSSP</td>
<td>aw</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSSP</td>
<td>ay</td>
<td>2</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSFBP</td>
<td>aa</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSFBP</td>
<td>al</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSFBP</td>
<td>av</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSFBP</td>
<td>aw</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>CSFBP</td>
<td>ay</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>aa</td>
<td>4</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>al</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>av</td>
<td>6</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>aw</td>
<td>6</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>ay</td>
<td>2</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>all</td>
<td>2</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SPMS</td>
<td>all</td>
<td>2</td>
<td>mixed</td>
<td></td>
</tr>
<tr>
<td>SFRA</td>
<td>ay</td>
<td>7</td>
<td>mixed</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Simulation parameters for each algorithm of Figure 5.35.
observation is that the accuracy gain of spatial plus spectral information algorithms over SPMS and CSP (purely spatial information algorithms) is very noticeable, which might indicate that it is much harder to achieve near perfect accuracy with only spatial information. Despite this fact, SPMS results improving over simple CSP show that more spatial information can be “squeezed” from the data.

Finally, although CSSSP produces very interesting results and is clearly the superior algorithm (as it can be applied almost blindly and is sure to achieve nearly optimal results), using accuracy as a single performance metric hides the fact that it requires much more computational effort. In fact, both CSSSP and SPMS, which rely on solving optimization problems and were implemented using “black box” cost functions, are the algorithms with the longest simulation times. This happened because deriving closed form expressions for the cost functions is far from trivial and doing so would only impact computational performance, which is outside the scope of this thesis. Additionally, the CSSP algorithm requires simulations to be run for many delay values. In this light, the previously presented results should be carefully analysed with respect to this, since accuracy and simulation time seem to trade-off.
5. Experimental results

Summary

This chapter presented the experimental results associated with work of this thesis, obtained from a set of simulations was performed using the algorithms presented in Chapter 3. First, an individual analysis of each algorithm was done, which included not only the accuracy results of the algorithm but also an interpretation of the parameters it generated and, where it was possible, a study of the algorithm's interaction with different spatial filter selection methods. Then, the effect of the frequency band selection procedure presented previously was studied and shown to be able to improve on the results of a CSP-based algorithm, followed by a comparison of classifiers. In the last section, the algorithms presented in this thesis were compared in terms of accuracy. The next chapter draws further conclusions, presents further work to perform over these results and wraps up the material for the thesis.
6 Conclusions

Contents

6.1 Result Review ................................................................. 90
6.2 Future work ................................................................. 91
6. Conclusions

This thesis sought to present a baseline knowledge in Electroencephalogram (EEG)-based Brain-Computer Interfaces (BCIs). To this end, it presented some basic neurophysiology concepts, described signal acquisition and a typical BCI’s structure and also explained the inner workings of the CSP algorithm, as well as of two pre-existing algorithms which are based on it, Common Spatio-Spectral Patterns (CSSP) and Common Sparse Spectral Spatial Patterns (CSSSSP). This knowledge can be used to provide a jumpstart to anyone initiating studies in the field of BCIs.

Additionally, it proposed two procedures to improve performance in BCIs, which deal with spatial filter selection and tuning of the pre-processing spectral filter, and three new algorithms, Combined Frequency Band Spatial Patterns (CFBSP), which builds on the filter tuning procedure previously proposed, Spatial Filter Residual Analysis (SFRA), which adds Auto-Regressive (AR) modelling to account for temporal structure in the signals, and Spatial Patterns for Maximal Separability (SPMS), which is based on a different way of computing spatial filters.

For simulating the above mentioned algorithms, a library of MATLAB functions was developed with emphasis on quick adaptation to new circumstances and algorithms, by defining core functions which perform basic operations and a normalized structure for a BCI simulator. This library was used to test all of the methods mentioned above and an analysis and comparison of their performance was made.

6.1 Result Review

The results presented in Chapter 5 confirmed that, despite some concerns of overfitting, CSP is a prime candidate for use in BCI applications, achieving over 70% accuracy for every subject in the chosen dataset. CSP’s results have, however, been improved upon by CSSP and CSSSSP, two algorithms presented in literature, which add temporal information and show improvements relative to CSP of 5% and 15%

One of the contributions of this thesis is the proposition of two new methods for selecting spatial filters, called the mixed and sorted methods (as opposed to the usual selection method, which we have called traditional). This was done because the traditional procedure doesn’t produce optimal results. Although not clearly superior, the proposed methods can achieve about 10% improved accuracy over traditional in the case of mixed and 8% in the case of sorted. The mixed method is particularly promising, since it's results on trials in the training set allow us to predict the performance on new trials better than the traditional method.

Another proposition is the frequency band selection method, which attempts to select the frequency band over which relevant brain activity is located. This information is then used to design the pre-processing filter and improve the SNR of the system's input. This procedure’s results show it to suffer heavily from overfitting but still provide an improvement over the usually employed frequency bands of 8%.
The three entirely new methods proposed, CFBSP, SFRA and SPMS, can also be used to improve upon the results of CSP. CFBSP, which uses a simplified version of the frequency selection method mentioned above to select two frequency bands which are then processed together, can improve results by 10%, while SFRA (using temporal filters derived from AR models) and SPMS (which computes spatial filters in a novel way and doesn’t rely on filter selection) provide a more moderate increase of 4%.

While comparing the results of using different classifiers with CSP, one could observe that LDA produces the best results, while NBC and SVM perform worse by 5%. Since not all algorithms could be tested with every classifier, these results are not representative and no definite claim can be made.

Finally, a comparison of all the presented algorithms was made, with CSSSP showing consistently superior results despite it’s sub-optimal filter selection procedure and high computational effort required.

6.2 Future work

Following the presented work, a few research directions can be taken. The first deals with the developed MATLAB library and consists of moving the current model, based on grouping algorithms using folders, to an object oriented approach with classes, which can be more suitable and easier to deploy and maintain.

Two of the proposed algorithms, CFBSP and SPMS, are novel ways to explore the EEG data and can be improved. CFBSP suffers from overfitting and this issue must be dealt with by, for example, adding crossvalidation to the frequency band selection. SPMS relies on an optimization problem which is computationally heavy and can thus benefit from alleviating some of the effort. This can be done by attempting to obtain an analytical expression for the cost function. Also, the SPMS algorithm only optimizes spatial filters and can be extended to include temporal or spectral information. This should lead to improved results.

The results from the spatial filter selection methods also lead us to believe that accuracy can be significantly improved by changing the way that filters are selected. Since the techniques outlined in this thesis are very simple, there is a great margin for improvement.

Finally, all of the algorithms showed overfitting. Although in some cases it is not severe, it is a generalized problem of BCI algorithms and should be addressed.
6. Conclusions
Bibliography


