Swiss Federal Institute of Technology

Semester Project
Design of a brain-actuated device for human-computer interaction

Student :	Abel Villca Roque	abel.villca@epfl.ch
Responsible Assistant :	Gary N. Garcia	gary.garcia@epfl.ch
Professor :	Touradj Ebrahimi	touradj.ebrahimi@epfl.ch

February 5, 2003

Acknowledgements

I would like to thank Mr. Touradj Ebrahimi for the advice and feedback he gave me during the semester.I would also like to thank Mr. Gary Garcia for the amount of time and work he invested in my project.

Abstract

A Brain Computer Interface (BCI) provides a new non-muscular channel for sending messages and commands to the external world.

Over the past few years, a BCI research program using electroencephalogram (EEG) signals has arisen at the Signal Processing Institute (SPI) at the Swiss Federal Institute of Technology, Lausanne.

This BCI was designed with four Operational Modes (OM): Visualization OM, Training without Feedback OM, Training with Feedback OM and the control OM.

In this project, a Control Operational Mode (Control OM) was designed and implemented. The main functionality of a control OM is described as follows: it receives EEG data taken from the user, interprets and classifies it as a Mental Activity Task defined in a previous Training Phase and finally executes a specific action using a rendering component, in this case the 3D platform.

In order to classify the signals with the control OM, the data received needs to be compared to a training model acquired during a Training Phase. This classification can be achieved using a classification technique known as Support Vector Machine (SVM). Within the scope of this project, a Support Vector Machine Library previously developed in Matlab was integrated into the BCI system.

The development of this 3D platform is also part of this project, such platform uses a 3D engine developed in C++ and MFC. Its task is to receive the Mental Activity Task and execute a predetermined action in a rendered 3D world.

Additionally, a Training with Feedback OM capable of using the SVM library and the 3D platform capabilities was developed. The Training with Feedback OM shows a visual cue to the use, treats the input data and shows a visual feedback to the user. The data flow in the Training with Feedback OM is similar to the one realized in the Control OM, the main difference lies in the visual cues display protocol.

In this report, a brief introduction to the BCI system architecture and its implementation is presented before the presentation of the main characteristics. Then, the experimental results of the system are exposed followed by the conclusions.

Contents

List of Figures

1 Introduction

The purpose of this report is to provide a brief but complete description of the implementation of a control module, the implementation of the 3D server and the integration of a Support Vector Machine library into the BCI project.

A Brain Computer Interface (BCI) allows an individual to communicate or control an external world without using the brain's normal output pathways of peripheral nerves and muscles. Messages and commands are expressed by electrophysiological phenomena or spontaneous EEG features. BCI depends on the interaction of two adaptive controllers, the user, who must maintain close correlation between his or her intention and these phenomena, and the BCI, which must translate the phenomena into device commands that accomplish the user's intent.[4]

The essential and central fact of BCI development and operation is that the BCI changes the electrophysiological signals (i.e. EEG) from mere reflections of the central nervous systems activity into messages and commands that act on the world. Basically, a BCI replaces nerves and muscles and the movements they produce with electrophysiological signals and the hardware and software that translate those signals into actions.

A very important point for the success of a BCI operation is that the user develop and maintain a new skill, a skill that consists not of proper muscle control but rather of proper control of specific electrophysiological signals.

Moreover, BCI research is an interdisciplinary problem involving the cooperation between neurobiology, psychology, engineering, mathematics and computer science.

An important issue for the functionality of a BCI system is the hardware and software implementation.

The goal of this project was the implementation of a control module integrating a translation algorithm (Support Vector Machine) and a 3D platform aimed at the user.

This implementation is essential for the BCI system since the control module represents the bridge between the data acquisition, the translation of this data into commands and the manifestation of these commands into actions.

In this way, a control module was designed and implemented so it can meet the control and time requirements of the BCI. Furthermore, a communication protocol between the SVM library and this control OM was designed and implemented in order to increase the functionality of the system.

A 3D platform based on a 3D engine was proposed for the representation of the BCI commands produced by the user. The 3D platform represents, in this case, the manifestation of the commands in a versatile, flexible and user-friendly environment. These platform is intended to be versatile and flexible so it can be used by other modules i.e. the feedback with training module.

2 Features of the BCI

2.1 EEG signals

An electroencephalogram (EEG) is the recording of the electrical potentials generated by the brain on the scalp, the potentials are in the order of 5-100

μV. Therefore, the EEG is recorded as a potential difference between a signal electrode on the scalp and a reference electrode (typically the ear). A sampling rate between 100-300 Hz are enough to capture the EEG frequency content. We used a sampling rate of 256 Hz. Figure 1 shows the aspect of an EEG signal.

Nevertheless, EEG are highly sensitive to noise and foreign artifacts like those produced by the eye movement or the electrical activity of scalp muscles.

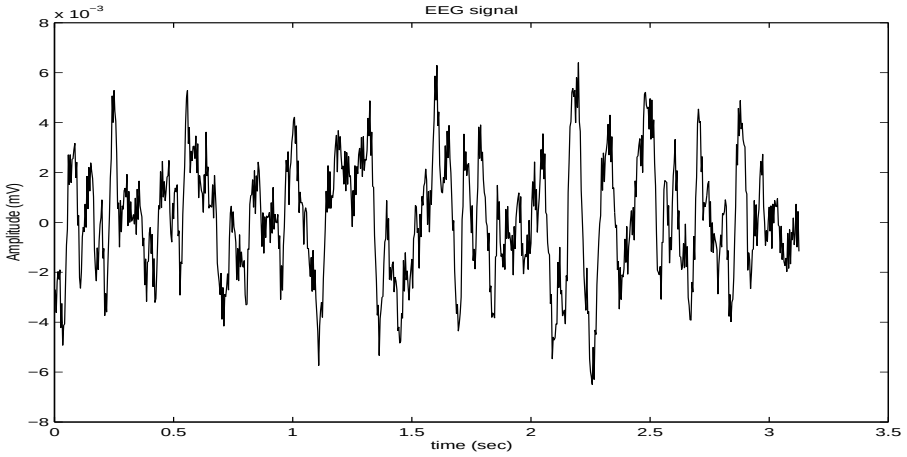

Figure 1: EEG Signal

2.2 BCI architecture

The BCI developed behaves like any communication or control system. It has an input (EEG signals from the user) and an output (an Action Command), it has components that translate input into output and a protocol that determines the timing of the operation.

The architecture of a BCI system is shown in figure 2 on page 5 and it consists of five main aspects.

1. Signal Acquisition
2. Signal processing: feature extraction
3. Signal processing: the translation algorithm
4. Output device

Signal acquisition EEG signals (see figure 1 are recorded from the user, amplified, digitized and transmitted to the Signal Processing module.

Signal processing: feature extraction Digitized signals are subjected to a variety of feature extraction procedures in order to choose representative features free of artifacts.

Signal processing: translation algorithm The translation algorithm translates these signal features into device control commands that carry out the

user's intent. In this project, a Support Vector Machine has been used to discern the device control commands from the signal features.

Output device The output device can be a computer screen, a projector or a pair of 3D glasses, it can also be any machine that could respond to the BCI device commands: a wheelchair, a pointer, etc. Also, this output can also be used as a feedback that the user's brain will use to improve or maintain the accuracy of the communication. In this project, the output device is implemented by a screen displaying an interactive 3D environment.

2.3 Operating protocol

When a new user first accesses the BCI, the algorithm needs to adapt to that users' signal features. An effective BCI needs also a second level of adaptation to reduce the impact of spontaneous variations: variations linked to time and day, hormonal levels, fatigue, illness, etc. The translation algorithm needs to adjust itself to these variations so it matches as closely as possible the user's current range of signal features values to the available range of device command values.

However, those two levels are not enough to address the central fact of effective BCI operation which is the dependence of the interaction between the brain and the BCI. A third level of adaptation will take into account the adaptive capacities of the user's brain so, in the long term, the brain will modify its signal features to improve the BCI operation. In short, the user needs to modulate its EEG signals too.

To cope with these aspects, an operating protocol needs to guide the BCI operations. The operating protocol defines the system behavior, the sequence and

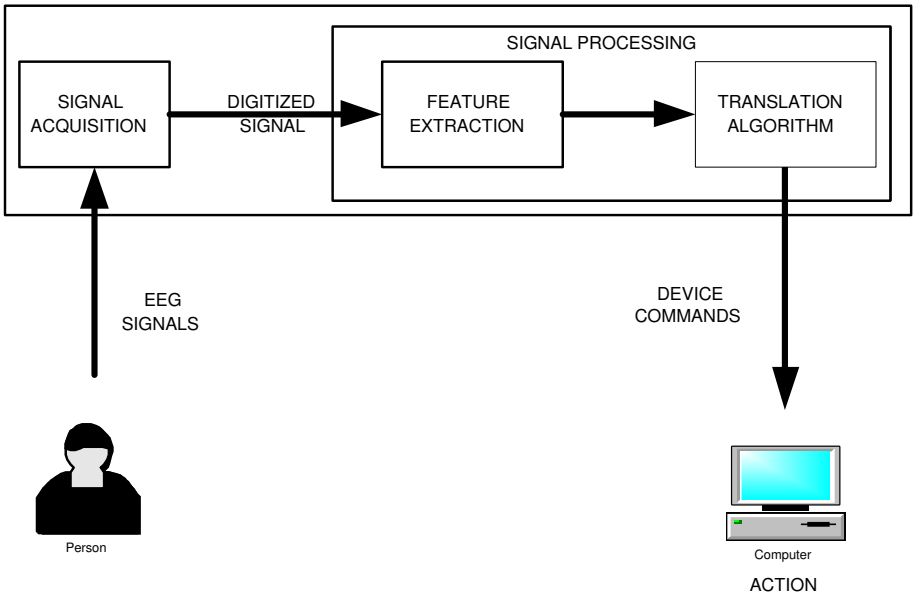

Figure 2: BCI Architecture

speed interactions between the user and the system and what kind of feedback is provided to the user.

As a result, the operating protocol consists of four Operational Modes:

Visualization. In this Operational Mode, the subject is able to see a visual representation of his EEG in real time. The objective of this OM is to calibrate the system and to familiarize the user to it.

Training without feedback. Visual cues are presented to the subject so he can perform predefined mental activities. The data is then recorded and analyzed and the computer can learn those EEG patterns associated with particular mental activities: a classification model is created using a support vector machine procedure. This process is carried out off-line.

Training with feedback. It has the same functionality as the Training without feedback Operating Model but with an additional characteristic: it provides feedback to the user. The user thus can adequate its signal so the system will be able to analyze this data and eventually enhance its classification model. The classification is based on the reference models of the mental activities created during the training without feedback OM.

Control. It is the most important entity of the BCI since the subject can start to control the system by producing mental activities for which the system was trained. Therefore, the control OM has the important task of generating the action command that will be sent to the 3D server. In order to implement the functionality of the BCI, the action command generation must be done in real time, it implies that the signal processing treatment needs to be achieved in real time also.

2.4 BCI implementation

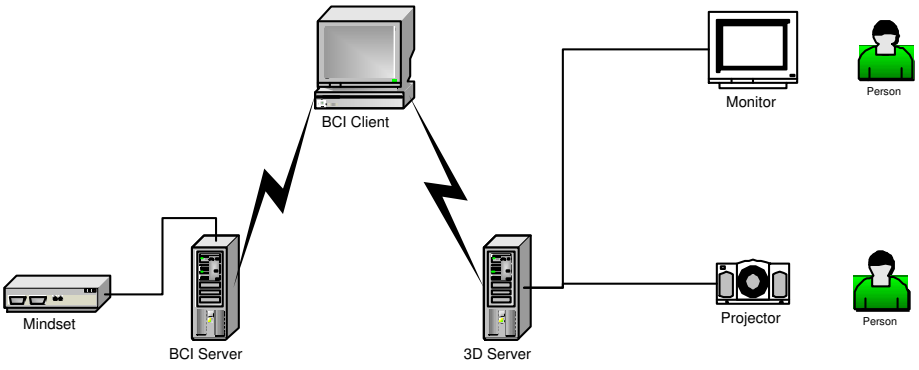

Figure 3: BCI Implementation

Figure 3 shows the implementation of a BCI system. It consists of five parts:

1. Mindset
2. BCI Server

3. BCI Client
4. 3D Server
5. Output device

Mindset EEG signals are acquired and digitalized by the Mindset MS-1000 device from Nolan Computer Systems. The main characteristics of this device and the data captured are detailed below:

- Channels: 16 differential input channels
- Resolution: 16 bit analog to digital converter
- Frequency: 256 samples/second/channel
- Input Range: 0 – 120 microvolts (μV) peak

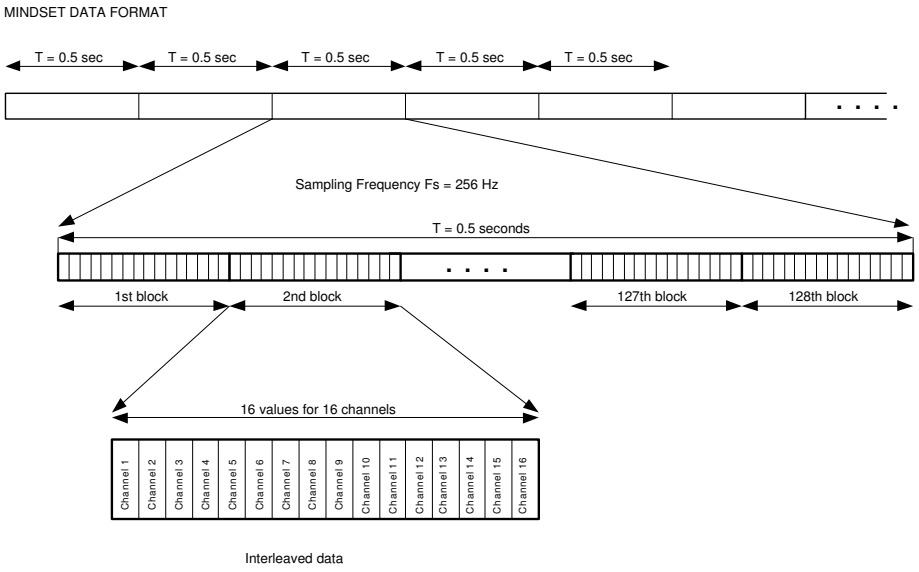

Figure 4: EEG signals data format

The signal received from the Mindset consists in a chunk of 2048 numbers corresponding to a period of half a second. The EEG data come from 16 channels at a frequency of 256 Hz (see figure 4).

BCI Server The BCIServer receives the data from the Mindset and send it to the BCIClient. The main task of the BCIServer is to provide the BCIClient with a periodic source of data according to the BCIClient requests.

The BCIServer is implemented in Java using the Mindset DLL library and a CORBA platform for the communication with the BCIClient.

BCI Client The heart of the BCI Implementation is the BCI Client since it contains the Signal Processing modules and the Operational Modes implementation.

As it is mentioned in section 2.2, the Signal Processing module implements the Feature Extraction and the Translation algorithm routines.

It also contains the Operational Modes development as detailed in 2.3

3D Server Implements the output device

3 Development and tests

This project faces three main issues:

1. Design and implementation of the control OM
2. Integration of a Support Vector Machine Library
3. Development of a 3D platform.

3.1 Design and implementation of the control OM

3.1.1 Overview

Basically, this module is in charge of the correct reception of EEG signals, the translation into Action Commands and the posterior dispatch of these Action Commands to the 3D server.

The Control OM is implemented in the BCI Client shown in figure 3, it is written in Java SDK 1.3.

3.1.2 How does it work?

Figure 5 describes the details of the control OM structure.

The control OM is started by the BCI administrator, who is the person responsible of the system. First, it enables the communication with the Support Vector Machine library implemented in Matlab. Then, it establishes the communication with the BCI Server and the 3D server. Once all the communication setup achieved, it will begin its Timer. The Timer is a sort of loop whose behavior is in accordance with a protocol better described in section 3.1.4. Each time the timer executes a loop, it performs the following operations:

- EEG data request. The control OM sends a request message to the BCI server
- Getting the EEG data from the BCI server. The BCI server sends a chunk of EEG data to the BCI Client, the data then is received by the control OM. Thus, each time the control OM receives data from the BCI server, it receives 16384 bytes of interleaved data among the 16 channels since each number is saved as double float.
- Sending EEG data to the SVM library. The data is sent to the SVM library.
- Receiving the Action Command from the SVM library. A response is received from the SVM module, this response is forwarded to the 3D server.

- Sending the Action Command to the 3D server.

All this process is repeated until the Timer stops. An important point to mention is the fact that the Timer behavior is based on which Operational Mode is being executed.

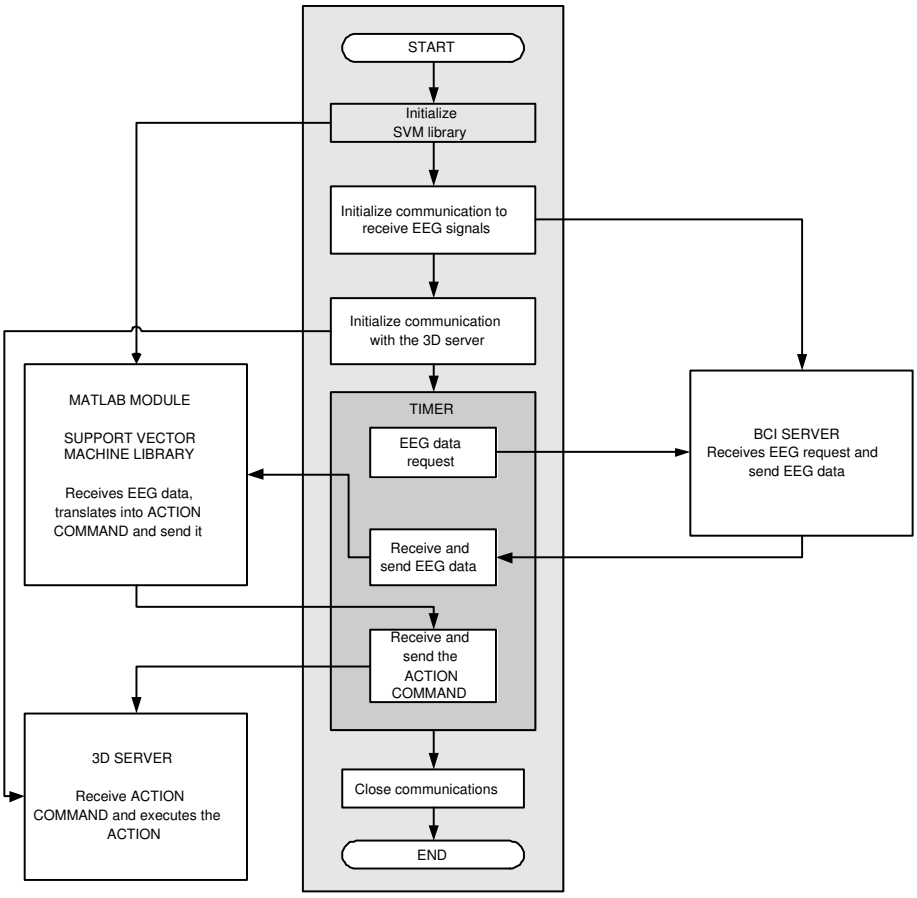

Figure 5: BCI Control structure

3.1.3 Software Engineering

The control OM was integrated into the BCI project according to the UML[1] diagram shown in fig 6

The software implementation is composed by four main operational classes:

- General Application class. Since this is the superclass of all the remaining applications, it implements common properties such as: the communication capabilities with the BCI server using a CORBA platform and the file management properties.

[1]The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

- Visualization class. As its name suggests, it implements the visualization OM. In brief, it receives EEG data from the BCI server and displays a visual representation on the screen.
- Training class. It performs the training without feedback OM, it displays visual cues aimed at the user according to the training protocol, receives EEG data and saves it into a file for an off-line analysis.
- Feedback Training class. This class receives the EEG data and implements the training with feedback OM. The main difference with the training class lies in the fact that this class sends the EEG data to the SVM module and displays the feedback to the user in real-time. The command action is transmitted then to the 3D server.
- Control class. The same operation as the feedback training class are performed except that the control class does not send any visual cues to the 3D server.

We have seen so far that the feedback training class and the control class are intimately linked.

A more detailed description of the UML models can be find at appendix B

3.1.4 Communication Protocol

Two protocols were implemented in this project, one for the BCI training with feedback and the other for the control protocol.

Training with feedback protocol

Figure 7 shows the details of the training with feedback protocol [1].

This protocol was implemented using a high-level thread API, the java.util.Timer class. This Timer fits well since the program must perform a task repeatedly.

In this way, the BCI administrator user determines which Mental Activity (MA) is going to be used in the training session.

First of all, the program begins the timer protocol. It consists of 4 slices of 5 minutes each. Each 5 minutes slice has its own period for recording along

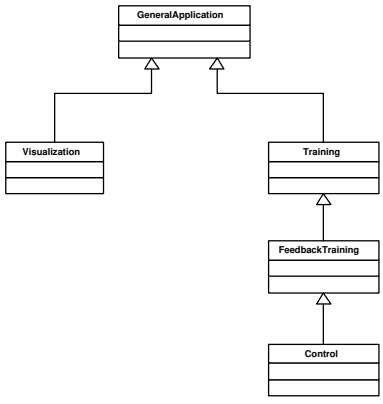

Figure 6: UML Outline

with break periods. The beginning of every period is determined by visual cues displayed in the 3D server. The Timer determines when the visual cues are displayed and when the data is recorded for future analysis.

Obviously, the data transmission needs to be controlled by this protocol as well. With this purpose, when all the data corresponding to a period is received, the whole chunk of data is sent to the SVM library and a response is received. This response corresponds to the Action Command or Feedback which is forwarded to the 3D server when the Timer gives the order to send it.

The visual cues are displayed in the 3D server according to the diagram shown in fig 7.

For that reason, a communication protocol was designed to make the difference between the visual cues, the action commands and the control commands. This protocol will be better explained in section 3.3.

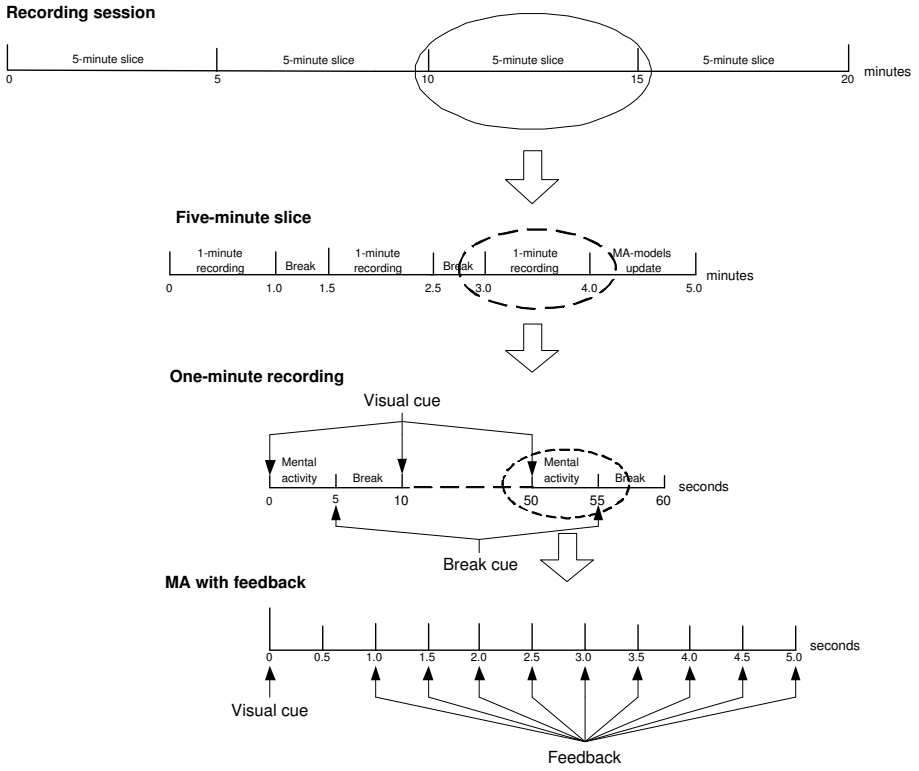

Figure 7: Training with feedback protocol

The Control protocol

The main difference between the control and the training with feedback protocol lies in the timing aspect. The control protocol doesn't implement the visual cues displays nor the recording operation.

Figure 8 shows that each half a second, the control OM submits an EEG data request to the 3D server. Once the EEG data received, the OM control is ready to send this data to the SVM module. This happens when the Timer sends the feedback instruction, the EEG data is thus sent to the SVM machine and

an Action Command is received; this command is forwarded to the 3D server according to the same communication protocol used to communicate with the 3D server by the BCI training with feedback module.

3.2 Integration of a Support Vector Machine Library

3.2.1 SVM Module Architecture

A Support Vector Machine was integrated into the BCI. The JMatLink package was used to enable the communication between the BCI Client program implemented in Java and the Support Vector Machine library implemented in MATLAB.

The main idea behind the integration of the SVM library and the BCI project was to allow an effective communication and to maintain their mutual independence at the same time.

An effective communication between the two modules means that the total time between the transmission of the EEG signal from the BCI Client and the reply from the SVM module needs to be less than the total time allowed to the Feedback reply when it runs in real-time.

The mutual independence is important for both modules. On the one hand every modification or development of the translation algorithm in the SVM module needs to be completely transparent to the BCI client. On the other hand any further development of the BCI client protocol doesn't need to concern the SVM module either.

Therefore, the following protocol was implemented:

The SVM library implemented in MATLAB is initialized at the beginning of the translation process as it is shown in figure 5.

The BCI client considers the SVM module library as a black box. Therefore, only three kind of messages are implemented in the communication protocol by the SVM module:

- Receive initialization message of the SVM module: It initializes the parameters needed for the SVM machine. The nature of these parameters will be explained later in section 3.2.2.
- Receive EEG data. A chunk of raw data is received from the BCI client as the input of the SVM "black box".
- Send Action Command. The Action Command is sent to the BCI Client after the translation.

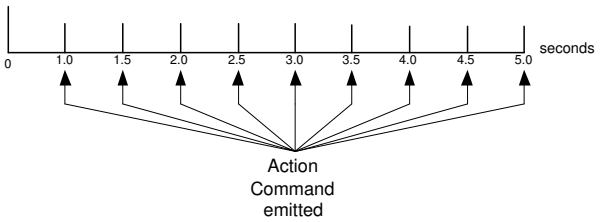

Figure 8: Control protocol

With these requirements, the communication is transparent and the system is flexible enough to make any modifications.

3.2.2 Support Vector Machines

Since the SVM theory is quite extensive and the theory has been developed widely, a description of the general aspects of the Support Vector Machine theory is presented at the appendix A.

Basically, an SVM is a parameterized function whose functional form is defined before training. Training an SVM requires a labelled training set, because the SVM will fit the function from a set of examples. The training set consist of a set of N examples. Each example consists of an input vector $\vec{x}$ and a label y_i, which describes whether the input vector is in a predefined category, each category corresponds to a defined Mental Action.

The input to the SVM module consists of:

- a kernel definition and its parameters
- input parameters
- one or more training vectors corresponding to the EEG data

Kernel definition The Radial Basis Function was chosen for the kernel definition:

$$K(\vec{x}, \vec{y}) = e^{-\gamma \cdot \|\vec{x} - \vec{y}\|^2}$$

where γ corresponds to the input parameter, $\vec{x}$ corresponds to the N dimensional pattern and $\vec{y}$ to the class labels.

Input parameters The SVM module also needs the definition of another parameter, this parameter is C governs the trade-off between the generalization and the training errors by penalizing the errors in the training set.

For these reason, the SVM module needs to be initialized with two parameters: γ and C.

Input data It is particularly important that the input data for the SVM is artifact free to ensure the adequacy of the training and classification process. Artifact free means that the chose features are not contaminated by electromyographic activity from muscles(i.e. when the subject blinks), electrooculography (EOG) or other artifacts.

3.2.3 Training phase

The training details are shown in figure 9 on page 14

Training data is needed to train the SVM and create an SVM model according. A Training data set is needed for each Mental Activity and create a Feature Vectors Matrix for this particular Mental Activity. Fig 9 shows the steps to follow to create an SVM model for two Mental Activities (MAs).

Before feeding the SVM algorithm itself with the input data coming from the BCI Client, it is necessary to normalize the data and create feature vectors.

These steps are achieved by the mean removal and the A-R model modules inside the SVM implementation. Since this kind of preprocessing is well known, we will not enter into the details of the preprocessing implementation.

Nevertheless, we need to know that at the end of the training phase, we will finish with a Feature Vectors Matrix for each MA. The SVM model is created using these matrices.

3.2.4 Classification phase

Figure 10 shows the details of the classification procedure on page 14

An SVM model must be created before entering into the Classification Phase. Once this model created for predefined Mental Activities, the system is able to

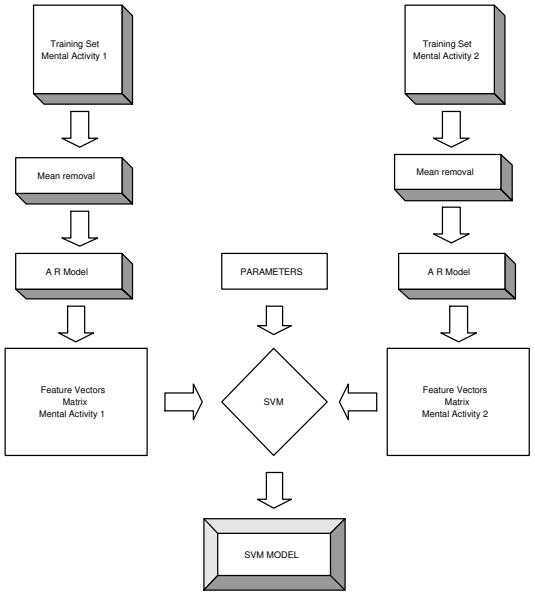

Figure 9: Support Vector Machine. Training diagram

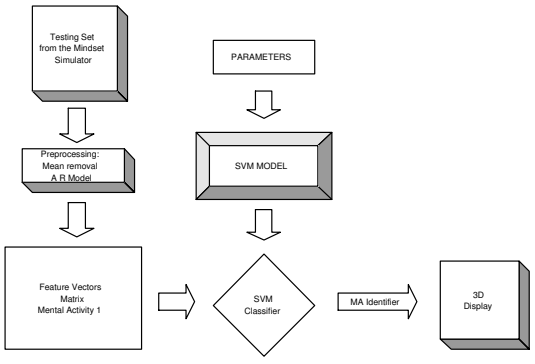

Figure 10: Support Vector Machine. Classification diagram

classify one feature vector at a time.

Each time the SVM module receive data from the BCI Client, this data corresponds to the amount of EEG data taken from the user in a period T. In our case, the period T is equal to 0.5 seconds.

The same preprocessing steps are repeated here to create the feature vector. Once this feature vector is calculated, it is fed directly into the SVM model which will return an Mental Activity Identifier to the BCI Client.

3.3 Development of a 3D platform

In order to create a friendly and versatile environment for the final user of the BCI, a 3D platform was developed. As we said before, this platform was implemented in the 3D server.

The 3D platform was developed in C++ using the Microsoft Foundation Classes (MFC) and the library of the Morfit 3D engine.

The 3D platform consists of three 3D worlds which implement the Action Commands and the Visual Cues received from the BCI client.

The only function of the 3D server is to receive orders and display its Action Commands. Those are the only actions allowed by the 3D server in order to obtain simplicity and polyvalence. Thus, all the timing is controlled directly from the BCI client.

3.3.1 3D worlds

Figure 11 shows the implemented worlds.

The Running Robot

This world consists of a robot moving in a platform. This environment can perform several tasks corresponding each to an Action Command: move forward, move right, move left, stop.

It also implements an intensity variable which implements a rewarding mechanism for the final user. For example, if the final user has been emitting a determined task for a predetermined time period, the robot would increase its speed. This kind of actions are regulated by the State Machine implemented in the Control OM module.

The Rotating Speaker

It consists in a speaker turning around one of the three spatial axis. It means that it is capable to display seven Action Commands: turn clockwise or counterclockwise around one of the three spatial axis (X,Y,Z) and stop.

The speaker can turn faster or slow down its rotation following the intensity variable command from the BCI Client.

The Flying Balloon

Finally, the flying balloon is the last world implemented. It can display for Action Commands: go forward, stop, go left and go right.

3.3.2 Communication protocol

In order to allow the communication with the 3D server, the following communication protocol was designed for each of the Operational Modes.

Figure 11: 3D worlds

A two bytes message is sent from the BCI Client to the 3D server to indicate whether the 3D server needs to display a visual cue or to execute a command action. The message needs to transmit the intensity as well.

The following format was chosen to implement this protocol:

First byte: Up to 256 possible values for the Visual cues and the Action Commands.

- 7 Action Commands: integer values from 97 till 103
- 7 Visual Cues: values from 48 till 57

Second byte: Up to 256 possible values for the intensity

- 10 levels of intensity: 0 to 9

The timing of the commands and visual cues messages depends on the task the BCI is accomplishing.

Training without feedback Only visual cues messages are sent from the BCI client.

Training with feedback The BCI Client sends Visual cues and action commands according to the Timer behavior.

Control Only Action Commands are sent.

4 Results and discussion

4.1 Mobility

Before going forward, let's define the final user as the individual who will give her EEG signals to the system, receive the feedback and control the 3d world. Moreover, the controller will be the person in charge of the experience.

With the configuration in figure 3, this system allows more freedom to the final user, the controller, the acquisition and display devices because of its networking capabilities. In our configuration, the 3D server, the BCI server and the Mindset along with the final user can be installed in a separate area whereas the controller can be in another room studying the signals without any disturbing interaction with the final user.

Obviously, this system can be implemented in one unique machine. In this case, this system needs to execute the BCI Server, the 3D server and the BCI Client at the same time. With this configuration, the performance of the system decrease and we experienced some delays in the responses.

4.2 BCI Server Simulation

We have seen so far that the EEG input data for the SVM module shouldn't be contaminated with artifacts. In this way, we used previously stored EEG data from one individual to test the real-time capabilities with the help of a Mindset simulator implemented in the BCI Server.

The Mindset simulator basically reads previously stored and clean data and sends it to the BCIClient. It has the same behavior as the BCI server reading

directly from the Mindset. For the BCIClient, nothing needs to be changed to receive the EEG data. This implementation is very useful when we want to test several sets of stored data.

4.3 Data Input

The data used to test this system has the following characteristics:

- Two sets of training data corresponding to two Mental Activities
- Two sets of testing data corresponding also to two Mental Activities
- All the input data correspond to artifacts-free EEG data from one individual

The two thirds of the whole data correspond to the testing set and the remaining data amount corresponds to the training one.

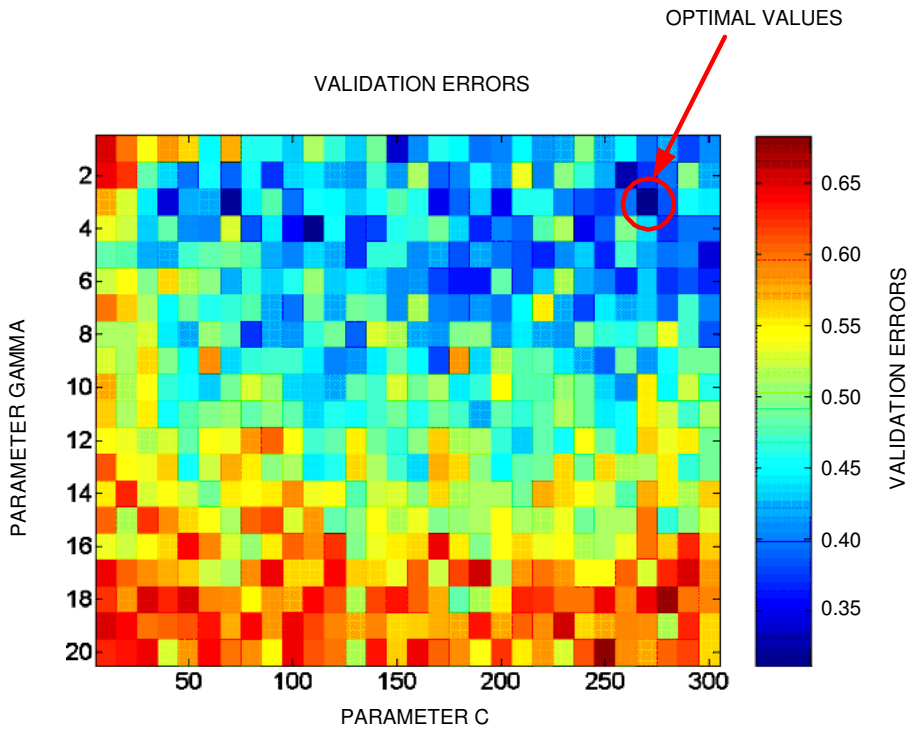

Figure 12: Validation error representation

4.4 SVM evaluation

The SVM model was created in an off-line mode using the training model described on figure 9. The SVM is fed with the training set described above.

Nevertheless, the correct parameters need to be found so the SVM model could be optimized. From the appendix A.6, we know that we need to minimize the following objective function:

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i Q_{ij}\alpha_j - \sum_{i=1}^{N}\alpha_i$$

subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} y_i\alpha_i = 0$.

The details of these developments can be found at the appendix.

Therefore, from section 3.2.2 we know that we need to feed the SVM with two parameters: γ and C.

Figure 12 shows the different values of the display error. These values are obtained using the training and the testing sets. We can observe that the best values were found when: $\gamma = 3$ and $C = 270$.

The control OM and the 3D performs the tasks they were supposed to do at the beginning of the project. The integration between the SVM and the BCI includes the integration of this library into the Control OM and the Training with Feedback OM modules.

However, the real-time issue is still something we need to deal with since the main obstacle was the effective suppression of the noise artifacts in real-time. That is the reason why we used clean and saved EEG data previously taken from an individual.

5 Conclusions

This section concludes this report with a summary of the major points presented in this project.

1. A Control Operational Mode was developed and integrated into the BCI. This mode allows the translation of the EEG data – received from the data acquisition device– into action commands, which are then forwarded to the display device. The core structure of the Control Operational Mode was also used with the Training with Feedback Operational Mode.

2. The translation algorithm was based on the Support Vector Machine technique which was implemented in MATLAB. The integration of the SVM library into the BCI system comes within the scope of this project. A communication protocol between the Matlab Library and the BCI was also implanted.

3. A 3D platform implementing three 3D environments was set up into the 3D server. These environments can execute several Mental Activities and the display of their corresponding visual cues.

4. Finally, these three parts are interconnected to each other since their activities are complementary and they complete the structure of the BCI. Further development is made easy because each part of the project is independent.

6 Appendix

A SVM theory

A.1 A practical consequence of learning theory

Support Vector theory is based on some simple ideas and provides clear intuition of what learning from examples is all about. It can also lead to high performances in practical applications.

The SV algorithm can be considered as lying at the intersection of learning theory and practice: for certain simple types of algorithms, statistical learning theory can identify rather precisely the factors that need to be taken into account to learn successfully. Nevertheless, real-world applications –like the BCI case– often mandate the use of more complex models and algorithms –such as neural networks– that are much harder to analyze theoretically. The SV algorithm achieves both. It constructs models that are complex enough: it contains a large class of neural nets, radial basis function (RBF) nets, and polynomial classifiers as special cases. Yet it is simple enough to be analyzed mathematically, because it can be shown to correspond to a linear method in a high-dimensional feature space nonlinearly related to input space. Moreover, even though we can think of it as a linear algorithm in a high-dimensional space. By the use of kernels, all necessary computations are performed directly in input space. This is the characteristic twist of SV methods.

A.2 Learning pattern recognition from examples

For pattern recognition, let's try to estimate a function $f : \Re^N \rightarrow \{\pm 1\}$ using training data –that is, N-dimensional patterns $\mathbf{x}_i$ and class labels y_i:

$$(\mathbf{x}_i, y_i), ..., (\mathbf{x}_l, y_l) \in \Re^N \times \{\pm 1\}$$

such that f will correctly classify new examples $(\mathbf{x}, y)$ –that is, $f(\mathbf{x}) = y$ for examples $(\mathbf{x}, y)$, which were generated from the same underlying probability distribution $P(\mathbf{x}, y)$ as the training data. If we put no restriction on the class of functions that we choose our estimate f from, however, even a function that does well on the training data –for example by satisfying $f(\mathbf{x}_i) = y_i$ (i going from $1, .., l$)– need not generalize well to unseen examples. Suppose we know nothing additional about f(for example about its smoothness). Then the values on the training patterns carry no information whatsoever about values on novel patterns. Hence learning is impossible, and minimizing the training error does not imply a small expected test error.

Statistical learning theory or VC(Vapnik-Chervonenkis) theory, shows that it is crucial to restrict the class of functions that the learning machine can implement to one with a capacity that is suitable for the among of available training data.

A.3 Hyperplane classifiers

To design learning algorithms, we thus must come up with a class of functions whose capacity can be computed. SV classifiers are based on the class of hyperplanes

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0, \mathbf{w} \in \Re^N, b \in R$$

corresponding to decision functions

$$f(\mathbf{x}) = sign((\mathbf{w} \cdot \mathbf{x}) + b)$$

We can show that the *optimal hyperplane*, defined as the one with the maximal margin of separation between the two classes, has the lowest capacity . It can be uniquely constructed by solving a constrained quadratic optimization problem whose solution $\mathbf{w}$ has an expansion $\mathbf{w} = \sum v_i \mathbf{x}_i$ in terms of a subset of training patterns that lie on the margin. These training patterns, called support vectors, carry all relevant information about the classification problem.

Omitting the details of the calculation, there is just one crucial property of the algorithm that we need to emphasize: both the quadratic programming problem and the final decision function $f(\mathbf{x}) = sign(\sum_i v_i(\mathbf{x} \cdot \mathbf{x}_i) + b)$ depend only on dot products between patterns. This is precisely what lets us generalize to the nonlinear case.

A.4 Features spaces and kernels

The basic idea of SV is to map the data into some other dot product space (called the *feature space*) F via a nonlinear map.

$$\Phi : \Re^N \rightarrow F$$

and perform the above linear algorithm in F. But this only requires the evaluation of the dot products.

$$k(\mathbf{x}, \mathbf{y}) := (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$$

Clearly, if F is high-dimensional, the right-hand side of the equation above will be very expensive to compute. In some cases however, there is a simple *kernel* k that can be evaluated efficiently. For instance, the polynomial kernel

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly d dimensions of $\Re^N$. For $d = 2$ and $\mathbf{x}$, $\mathbf{y} \in \Re^2$, for example, we have:

$$(\mathbf{x}\cdot\mathbf{y})^2 = \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 = \left(\begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{pmatrix} \right) = (\Phi(\mathbf{x})\cdot\Phi(\mathbf{y}))$$

defining $\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. More generally, we can prove that for every kernel that gives rise to a positive matrix $(k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$, we can construct a map Φ such that $k(\mathbf{x}, \mathbf{y}) := (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ holds.

Very typical kernels are the radial basis function (RBF) such as

$$k(\mathbf{x}, \mathbf{y}) = e^{(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})}$$

and sigmoid kernels (with gain κ and offset Θ)

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$$

A.5 Towards Support Vector Machine

We now have all the tools to construct nonlinear classifiers. To this end, we substitute $\Phi(\mathbf{x}_i)$, and perform the optimal hyperplane algorithm in F. Because we are using kernels, we will thus end up with nonlinear decision function of the form

$$f(\mathbf{x}) = sign(\sum_{i=1}^{l} v_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b)$$

The parameters v_i are computed as the solution of a quadratic programming problem.

In input space, the hyperplane corresponds to a nonlinear decision function whose form is determined by the kernel.

The algorithm just described has the following properties:

- It is based on statistical learning theory
- It is practical (as it reduces to a quadratic programming problem with a unique solution), and
- It contains a number of more or less heuristic algorithms as special cases: by the choice of different kernel functions we obtain different architectures such as polynomial classifiers, RBF classifiers and three-layer neural nets

The most important restriction up to now has been that we were only considering the case of classification. However, a generalization to regression estimation –that is, to $yin\Re$, can be given. In this case, the algorithm tries to construct a linear function in the feature space such that the training points lie within a distance $\epsilon in\Re$. Similar to the pattern-recognition case, we can write this as a quadratic programming problem in terms of kernels. The nonlinear regression estimate takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{l} v_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$$

To apply the algorithm, we either specify ϵ a priori, or we specify an upper bound on the fraction of training points allowed to lie outside of a distance ϵ from the regression estimate (asymptotically, the number of SVs) and the corresponding ϵ is computed automatically.

A.6 SVM implementation issues

SVM have proven to be very effective in real-world classification tasks. An SVM is parameterized function whose functional form is defined before training. Training an SVM requires a labeled training set, because the SVM will fit the function from a set of N examples. Each example consists of an input vector, $\mathbf{x}_i$, and a label, y_i, which describes whether the input vector is in a predefined category. There are N free parameters in an SVM trained with N examples. These parameters are called α_i. To find these parameters, the quadratic programming (QP) problem must be solved:

minimize:

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i Q_{ij}\alpha_j - \sum_{i=1}^{N}\alpha_i$$

subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} y_i\alpha_i = 0$.

where Q is an $N \times N$ matrix that depends on the training inputs $\mathbf{x}_i$, the labels y_i, and the functional form of the SVM. We call this problem *quadratic programming* because the function to be minimized (called the *objective function* depends on the α_i quadratically, while α_i only appears linearly in the constraints. The definition and applications of $\mathbf{x}_i, y_i, \alpha_i$,and Q can be found in [2].

B UML design

B.1 Control module

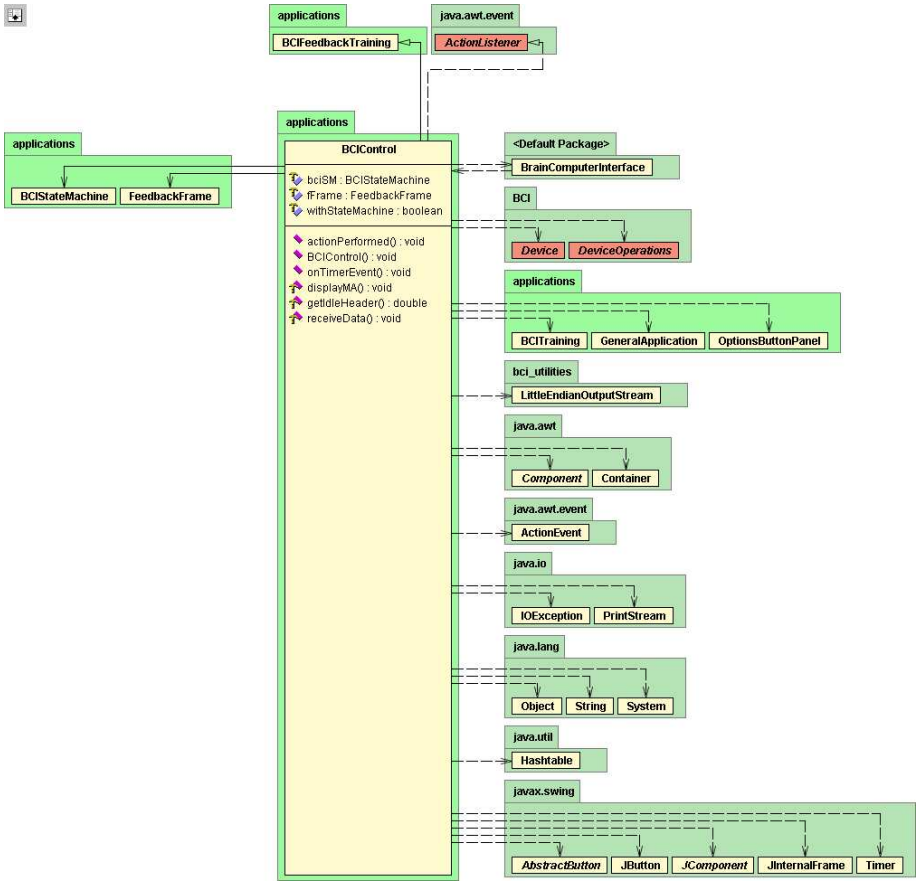

Figure 13: BCI Control UML diagram

B.2 BCI training module

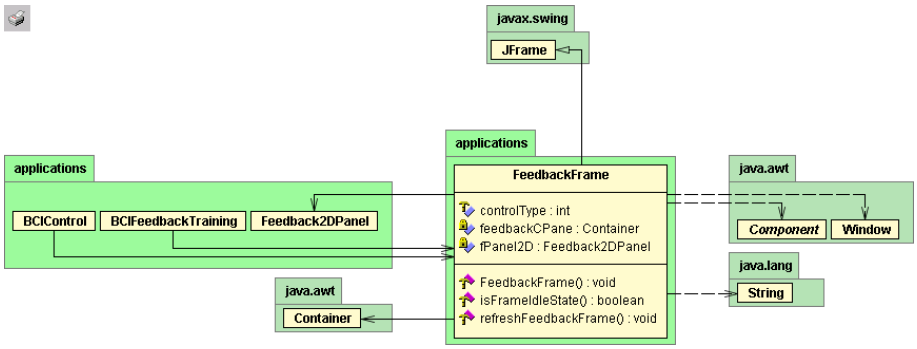

Figure 14: BCI Training with Feedback UML diagram

References

[1] Gary N. Garcia, T. Ebrahimi and J.-M. Vesin, "Joint time-frequency-space classification of EEG in a Brain-Computer interface application"

[2] Christopher J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition" Data Mining and Knowledge Discovery, Vol 2 No 2,1998, pp. 121-167.

[3] G. Garcia, T. Ebrahimi, J.-M. Vesin and A. Villca, "Direct Brain-Computer Communication with User Rewarding Mechanism, Submitted to the IEEE International Conference on Information Theory ISIT 2003.

[4] Jonathan R. Wolpaw, Niels Birbaumer, Dennis J. McFarland, Gert Pfurtscheller, Theresa M. Vaughan, "Brain-computer interfaces for communication and control" Clinical Neurophysiology 113 ,2002, pp. 767-791

[5] G. Garcia and T. Ebrahimi, "Time-frequency-space kernel for single EEG-trial classification", Nordic Signal Proc. NORSIG, 2002.

[6] G. Garcia, T. Ebrahimi, J-M. Vesin, "Classification of EEG signals in the ambiguity domain for brain computer interface applications," IEEE Int. Conf. Dig. Signal Proc., vol. 1, pp. 301-305, 2002.

[7] Gary N. Garcia, T. Ebrahimi and J.-M. Vesin, "Correlative exploration of EEG signals for direct Brain-Computer Communication", Submitted to the IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2003.

[8] T. Ebrahimi, J.-M. Vesin, "Brain Computer Interfaces for multimedia communication", IEEE Signal Processing Magazine January 2003.

[9] Gary N. Garcia, T. Ebrahimi and J.-M. Vesin, "A direct brain-computer communication device with user rewarding mechanism", Submitted to the IEEE International Conference on Information Theory ISIT 2003.

[10] Marti A. Hearst "Support Vector Machines", IEEE Intelligent Systems, July/August 1998, pp.18-28