Introduction to NeuroComm: a Platform for Developing Real-Time EEG-based Brain-Computer Interface Applications

Chauchu Wang, Haihong Zhang, Kok Soon Phua, Tran Huy Dat, Cuntai Guan
Institute for Infocomm Research, Singapore 119613
E-mail: {ccwang, hhzhang, ksphua, hdtran, ctguan}@i2r.a-star.edu.sg

Abstract—NeuroComm is a platform to develop real time Brain Computer Interface (BCI) applications. This paper introduces the basic modules of this platform and discusses some implementation issues. With a user management module, NeuroComm is user friendly, suitable for multiple users. With flexible configuration files and signal processing algorithm libraries, it is easy to integrate multiple BCI applications into one system. It is also a flexible tool for BCI research.

I. INTRODUCTION

Brain Computer Interface (BCI) assists patients with severe physical disabilities to communicate with the outside world and control assistive devices. In recent years, there are increasing interests in the area of BCI research and applications. Among them, BC12000 [1] provides a general purpose research and development system. The Though Translation Device (TTD) [2] is another system. However when building real time BCI system, we realize that user friendliness is crucial for the acceptance of the system by patients. Moreover, a general-purpose platform facilitates the building of various BCI applications. To achieve this goal, with reference to our previous works [3-9], we designed and implemented a platform NeuroComm, which is introduced in this paper, together with some implementation issues.

II. SYSTEM DESIGN

The structure of NeuroComm is shown in Fig1. There are four main modules: application configuration, application interface, data acquisition and signal processing. The following sections will discuss all these modules.

A. Application Configuration

This module maintains a global configuration file which defines system level configurations, such as the current user, configuration files and BCI applications in the system. The console window shown in Fig2 allows users to enroll new users with the required information and select the current user and amplifier. The rest of the buttons are BCI applications that are included in the system. The message window under the menu bar displays debugging and processing messages.

BCI applications in the system and the configuration data for each application. The console window is shown in Fig2.

1) User Management

Fig2 shows user name “Haihong” in the window title (as subject). The application console enrolls new user with name, gender, age and other relevant information as required by the respective BCI application. When a new user is enrolled, a
new entry is created in the user database. System operator needs to set a current user. When an application starts, system loads in all the user specific configuration data.

With user management functionality, user’s configuration data and working environment settings are kept in system, not affected by other users’ usage or system restarting.

2) Configuration Files
NeuroComm has a set of various configuration files:
- EEG amplifier configurations of amplifier-specific parameters, such as channel names and sampling rates, and information needed to create amplifier objects.
- Application configurations of application specific parameters, such as task name, GUI settings etc.
- Algorithm configurations of signal processing functions, such as filter design and transform matrix etc.

3) Application Configuration
The global configuration file has an entry listing all BCI applications for the system which are also shown as buttons in the toolbar of the application console. To configure an application into this system, follow the following steps:
- Adds an application name into the “AppNames” list;
- Creates a section with the name of the application;
- Adds a “TypeName” entry with a type in the system type library as the value;
- Adds a “CfgFiles” entry with a value of a list of all app-configuration files that the application needs;
- Adds one entry for each app-configuration file, with a value of a list of consisting system configurations.

When the application is started for a new user, system creates a working environment for this user with the necessary configurations. For an old user, if any configuration file is missed, system recreates a new one. The configuration value can be modified later without affecting any other user.

B. Application Interface Module
This module consists of three components as listed below:

1) GUI component
It displays the application specific graphic interface, such as visual stimulus, graphic feedback etc., to the user.

2) System control component
It controls the application process flow, such as starting, initialization, checking for processing results and stopping the application’s execution, etc.

3) Device control component
It translates the signal processing result into control signal and controls outside devices. Here is a list of such controls:
- Cursor movement on the screen;
- Input of a letter into a word speller window;
- Selection of TV channel;
- Turning of the switch of an electrical device.

C. Data Acquisition
This module reads EEG data from EEG amplifier and passes to signal processing module. It also receives stimulus codes from application module to construct event markers in the EEG data. It saves both the EEG data and event markers to the hard disk for off-line analysis.

There are many different types of EEG amplifiers with different specifications. Some of them are listed in Table 1. A common interface has been designed, with each type of amplifier having its own implementation and communication mechanism with the hardware.

<table>
<thead>
<tr>
<th>Amplifiers</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuamps</td>
<td>40 channels, USB port as power source and interface to PC, with EEG cap, Data ACQ though TCP/IP connection to neuroscan server. Trigger input port.</td>
</tr>
<tr>
<td>SynAmps2</td>
<td>66 channels, separate power, USB port interface to PC, used with an EEG cap, Data AC using TCP/IP connection to neuroscan server. Trigger input port.</td>
</tr>
<tr>
<td>BioRadio150</td>
<td>8 channels, wireless, USB port as power source and data interface to PC, c++ APIs for online ACQ; user unit powered on 2 AA batteries which can run for 24 hours. No trigger input port.</td>
</tr>
<tr>
<td>g.USBamp</td>
<td>16 channels, power on AC, USB interface to PC, c++ API for online ACQ. No trigger input port.</td>
</tr>
</tbody>
</table>

All information here can be found from the manufacturer’s webpage.

Fig3 shows two configuration systems, each one having its own advantages. The selection of amplifier depends on the requirements of the specific application.

![System A](Image1.png)  ![System B](Image2.png)

Fig. 3. Two EEG configuration systems. System A uses Nuamps, an EEG cap and a computer; while system B uses BioRadio150, and head band with electrodes attached and a computer. The advantage of system B is that the user is not bound to the computer, he/she can move freely because the user unit is separated from the computer unit, and they communicate with each other through wireless radio. On the other hand, system A has more EEG channels and usually higher resolution.

Two virtual amplifiers are built in this module: one is Simulator which simulates a real amplifier by reading data from hard disk file; another one is Live-Simulator, same as Simulator except that it follows the speed of a real amplifier while reading EEG data. These virtual amplifiers are useful for offline study and debugging, with them we can repeat the
D. Signal Processing

This module processes the EEG data, extracts signal features, classifies these features using pattern recognition and machine learning algorithms ([4-9]) and passes the processing results to the application module to be translated into device control signal.

III. SOME IMPLEMENTATION ISSUES

We implemented the system in .net C++ and C#. The data ACQ module and the signal processing module are written in C++, because most EEG amplifiers come with Windows drivers and C++ interfaces, and most signal processing algorithms are written in C++. On the other hand, the interface module is in C# because of its easiness in GUI design. In this section, we discuss some problems we faced and the methods we used to solve them. In [3], we have discussed some problems such as the synchronization between different modules and the accurate timing problems. Here, we will discuss some other problems.

A. Event markers

Event markers are used to mark special events such as user responses and system cues (visual or auditory) during data collection. They are integer values forming a special channel in the EEG data. At signal processing stage the EEG data can be segmented and labeled using these event markers.

Some EEG amplifier has hardware triggering mechanism to generate event markers. For example, Nuamps, SynAmp2 from CompuMedics have trigger input ports which interface with a PC with a 25-pin parallel port. However we have a problem with Nuamps. We found that when the stimulus is sent at 40Hz or higher frequency (sending events every 25 millisecond or less), some events are lost.

Some amplifiers, such as BioRadio150 from CleveMed, g.USBamp from g.TEC, don’t have hardware trigger inputs. We need to either modify one EEG channel to be a trigger input channel, or use software to generate event markers.

1) Detection and recovery of lost hardware event markers

For amplifiers with hardware trigger input ports, a mechanism has been used to detect and recover lost event markers. By keeping a queue of stimulus codes received from amplifier module, system checks the event markers read from the EEG amplifier. Whenever a mismatch found, a lost event mark is identified. The error processing program is designed to recover this type of error by inserting the stimulus code into the event marker’s channel or report an error and interrupt the system execution, according to the system configuration.

2) Software generation of event markers

For EEG amplifiers without trigger input, an algorithm has been designed to generate event markers. A simple implementation is that when a stimulus code is received, the system inserts it into event marker’s queue. This method may encounter problem if accurate timing of triggering is required, because the EEG data is read in blocks; each block usually consists of multiple samples. For example, for an amplifier with 250Hz sampling rates and block size of 10 samples, one block will be 40 milliseconds. If the stimulus codes are sent every 30 milliseconds, the generated event marker queue will not be accurate; as illustrated in Fig 4.

A more accurate algorithm is using the time duration between consecutive stimulus codes to determine the number of samples between the relevant event markers.

B. Communication between modules

There are two ways for modules written in C# to communicate with modules written in c++, the first one is to create a wrapper .net c++ (managed c++) project which can access all c++ classes, and another way is to create a dynamic link library to export c++ functions. We have tried the two methods and found that the later one is much easier as we need not combine managed and unmanaged c++ classes together, which might cause linking problems.

1) C# calls C++ functions

Two steps are to be followed for this kind of interfaces:

- Create unmanaged C++ dynamic link library to expert all the C++ functions to be called by c#.
- Create proxy classes in C# to represent modules written in C++, defining all the interfaces that C# to call.

2) C++ calls C# functions

The application operator needs to check the intermediate data in order to adjust parameters. This requires modules written in c++ to pass data to C# for display. It is not practical for C# to call C++ functions to get the result for display because the application interface does not know when the data are ready; chances are that it may miss some data. Sometimes C++ modules also need to call C# functions to output results.

Fortunately C# has a delegate type which can be passed to C++ as a function pointer.

- C# defines a new delegate type to declare the parameters and return type of the function to be called
by the C++, creates an object and passes it to C++ as a function pointer;
- C++ calls the function referenced by the pointer to pass data to application interface in C#;
- C# displays intermediate results.

Fig 5 shows a window displays raw EEG data, extracted features and processing results. All data are passed to C# from the C++ implemented modules: data acquisition and signal processing modules.

C. Debugging message and system logs

In fig2, we can see a message window that displays system message such as debugging information and processing statuses. Such messages are also saved in a system log file for offline inspection. Our mechanism makes this display window transparent to all modules; all they need to do is just to write debug and process messages to the standard output stream. We have used the C++ pipeline and file redirection methods to capture standard outputs both from C++ and C# modules and redirected to the message window and the log file.

This mechanism helps to achieve the independences of different modules.

IV. CONCLUSION

NeuroComm is a platform that is flexible that applications can be included into one system using configuration files. With this platform, we have created a number of applications. Fig 6 shows four BCI applications with configuration console shown as Fig 2.

REFERENCES


Fig.5. A C# window in application interface. Top-left is raw EEG signal, top-right is intermediate processing result; bottom left shows the signal processing result. The data of all these display are passed by C++ functions from the data ACQ and signal processing module.

Fig. 6. BCI applications integrated into one system. a): soccer game, user coordinates the mu-beta rhythms in his EEG recorded from the motor cortex area of the human brain to control the movement of the soccer to the goal. b): maze solving game. User controls the movement of a bird along the maze paths to eat out all the seeds (pink dots). Three eagles (blue, yellow and brown) randomly walk along the paths. User needs to control the bird to avoid the eagles. c): A virtual TV controller. User selects a button by gazing on it while all buttons are flashing, dy: a brain controlled rehabilitation process. The user actively chooses a rehabilitation process through motor imagination.