Developing Wearable Bio-Feedback Systems: a General Purpose Platform

Luigi Bianchi, Fabio Babiloni, Febo Cincotti, Marco Arrivas, Patrizio Bollero, Maria Grazia Marciani

Abstract—Microprocessors, even those in PocketPCs, have adequate power for many real-time biofeedback applications for disabled people. This power allows design of portable or wearable devices that are smaller and lighter, and that have longer battery life compared to notebook-based systems. In this paper, we discuss a general-purpose hardware/software (HW/SW) solution based on industrial or consumer devices and a C++ framework. Its flexibility and modularity make it adaptable to a wide range of situations. Moreover, its design minimizes system requirements and programming effort, thus allowing efficient systems to be built quickly and easily. Our design has been used to build two Brain Computer Interface (BCI) systems that were easily ported from the Win32 platform.

Index term: Biofeedback, Brain-Computer Interface, BF++, Wearable devices, Augmentative communication.

I. INTRODUCTION

There is increasing need to develop wearable and inexpensive Cognitive Bio-Feedback (CBF) systems, small devices with long battery life that can have a wide range of applications for disabled people inside or outside the home. In designing a CBF application, it is necessary to identify specific voluntarily controlled activities (VCA), such as arm movements or mental tasks and link them to specific actions that will be performed by a dedicated machine. The role of the CBF system is to translate the VCAs into proper commands. To achieve this goal, the subject executes the VCAs while the system acquires and processes a biological signal that represents them (EMG or EEG). If the system is able to recognize which VCA generated the acquired signals - by means of special modules called classifiers - it can perform the action that has been previously associated with it. In this way, if the system is able to recognize four different VCAs, it can, for example, drive the cursor movement on a PC screen.

Many flexible systems are described in the BCI literature: Matlab-based systems [1] are excellent for rapid prototyping, but tend to be resource-consuming and quite expensive; BCI 2000 [2] is a highly modular system that requires less processing power, but is available only on the Win32 platform; P300-based systems such as [3] usually require several devices, are often cumbersome, and need a real-time Operating System (OS) since they require greater timing accuracy than EEG-based systems.

Real-time and embedded systems offer a much better platform to build wearable and inexpensive CBF systems, provided that their limited processor and memory resources are efficiently utilized.

The BF++ framework [4] is a software package that supports the creation of a wide range of CBF systems. Inspired by BCI systems described in the literature [5], it was designed to scale across different platforms and to optimally use all of the system resources. Even using low-power CPUs or CPU boards, it is possible to build portable and wearable BCI devices. In this paper, we describe a platform for such applications. It uses Windows CE, a small, configurable, real-time, feature-rich, 32-bit OS.

Many small CPU boards use the Windows CE OS with typical power requirements much lower than those of notebooks so that battery life can be up to one order of magnitude longer. Moreover, these boards are often equipped with many standard communication ports that can be used to connect acquisition devices or other peripherals. For these reasons, such devices represent an excellent platform for portable and wearable Bio-Feedback (BF) systems.

In addition to presenting our general-purpose hardware and software solution for such applications, we present two BCI systems, which were easily ported from the Win32 platform. This also suggests that it will be easy to migrate to newer platforms whenever smaller and more powerful devices will be available and that in some cases it is possible to build complete CBF applications with single chip PC or micro controllers.

II. MATERIALS AND METHODS

A. Hardware

Three different CPU boards from RLC Enterprise Inc. (CE-Plus, CE-Minus and ARM-Plus [6]) and a commercial Pocket PC (Compaq iPAQ H3650) were used. Their characteristics are compared in Table 1.

The CE-Minus board is the smallest (114/62/25 mm.). It requires only 350 mA for low-power mode, making it suitable for many applications that need long battery life (24 hours) and portability. The CE-Plus provides these same characteristics, but it also has a built-in display and six A/D...
12-bit converters. The ARM-Plus model provides the best processing capabilities and the best display quality making it suitable for attachment to an extended device such as a wheelchair. The CE-Plus and the ARM Plus are more cumbersome than the consumer PocketPC, but they provide many built-in connection opportunities (RS-232, USB, PCMCIA, Modem, Audio, A/D converters etc.) that can be used either to acquire biological signals or to drive external devices such as wheelchairs or other remote controllers.

### TABLE 1

<table>
<thead>
<tr>
<th></th>
<th>CE-Minus</th>
<th>CE-Plus</th>
<th>ARM-Plus</th>
<th>COMPAQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AMD Elan SC-400 x86</td>
<td>AMD400 Strong ARM SA-1110</td>
<td>Strong ARM SA-1110</td>
<td></td>
</tr>
<tr>
<td>Clock Speed [MHz]</td>
<td>100</td>
<td>100</td>
<td>206</td>
<td>206</td>
</tr>
<tr>
<td>DRAM Memory</td>
<td>12 MB</td>
<td>12 MB</td>
<td>32 MB</td>
<td>32 MB</td>
</tr>
<tr>
<td>RS232</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>Optional</td>
</tr>
<tr>
<td>USB</td>
<td>NO</td>
<td>1</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Display resolution</td>
<td>(Opt.) 320·240</td>
<td>320·240</td>
<td>640·480</td>
<td>240·320</td>
</tr>
<tr>
<td>Display colours</td>
<td>(Opt.) 16</td>
<td>16</td>
<td>65536</td>
<td>4096</td>
</tr>
<tr>
<td>Touch screen</td>
<td>Optional</td>
<td>YES</td>
<td>YES</td>
<td>Yes</td>
</tr>
<tr>
<td>PCMCIA</td>
<td>Optional</td>
<td>YES</td>
<td>YES</td>
<td>Optional</td>
</tr>
<tr>
<td>A/D channels</td>
<td>6</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A/D resolution</td>
<td>-</td>
<td>12 bits</td>
<td>10 bits</td>
<td>-</td>
</tr>
<tr>
<td>Size (W/H/D) [mm.]</td>
<td>114/62/25</td>
<td>195/116/50</td>
<td>200/120/50</td>
<td>83/129/15</td>
</tr>
<tr>
<td>Min / Typ Power</td>
<td>350 / 500</td>
<td>350 / 1100</td>
<td>1600</td>
<td>-</td>
</tr>
<tr>
<td>Audio In / Out</td>
<td>-</td>
<td>-</td>
<td>Mono/mono</td>
<td>Mono/mono</td>
</tr>
</tbody>
</table>

### B. Software

The BF++ (Bio-Feedback in C++) framework was used to provide all the typical functionalities of a CBF application. By grouping all the aspects that are common to all the CBF systems implementations, it is possible to use a “top-down” design. This approach offers much greater flexibility and allows the creation of a framework that can dramatically speed up the realization of several systems.

As illustrated in Fig. 1, our CBF system is composed of six main software modules:

1) The **Acquisition** module behaves like a driver and is responsible for acquiring the biological data and transmitting them to the **Kernel** module;

2) The **Kernel** module is the most important and is responsible for fulfilling all the other module requests. It is the concertmaster of the CBF application and sets the timing of the whole system, feeding the classifiers whenever new data are available, evaluating system performance, and driving all the other modules. The **Kernel** is composed of several sub-modules, each responsible for implementing a single aspect of a CBF application, such as the structure of a trial, the classification rule, the processing to be performed or the flowchart of the different operating modalities (e.g. training, testing);

3) The **Feedback Rule** module is responsible for defining the way in which the output of the classifiers should be used;

4) The **Patient Feedback** module is directly driven by the **Feedback Rule** module and provides the feedback to the user (e.g., an acoustic signal, a cursor movement on a PC screen);

5) The **Persistent Storage** module furnishes the required routines to load and save the data, either the acquired data or the processed data. It uses the XML technology to allow cross-platform portability among different platforms;

6) The **Operator User Interface** module is responsible for configuring the system, loading and saving the data, monitoring the system state, and setting other module parameters (operating modalities, acquisition and classifiers parameters, etc.);

These modules interact through well-defined software interfaces, so that it is possible to modify a module internally without affecting the others. Reusable components for the realization of CBF systems are provided. The whole BF++ package is written in ANSI C++ in order to make it portable across different hardware and software platforms, ranging from Workstations to Pocket PCs and from Desktop PCs to Smart Phones, regardless of the operating system. It has been used to build BCI systems [7] under the Windows platform, and it has been recompiled for offline analysis or demonstrative purpose under the Linux (kernel 2.4), the Windows CE 3.0, 4.0 (.NET) and the SmartPhone 2002 platforms.

The BF++ framework makes no assumptions about the usable biological signals and several can be used simultaneously in order to use all the patient’s residuals capabilities (e.g., EEG, EMG, voice, facial expression, etc.) — a key issue to guarantee the higher communication bandwidth. The main advantages of BF++ are the minimization of the programming effort, scalability, cross-platform portability, independence from the choice of biological signal, modularity, efficiency, and integration with existing devices and systems. More details about the BF++ are available in the literature [4] and electronically [8].
In this implementation, the BF++ is recompiled under the Windows CE 3.0 OS using the Microsoft Embedded Visual C++ 3.0 (EVC).

Following the BF++ model, a generic CBF application provides 4 different main modalities, even if they can be extended freely: training, testing, setup, and running. Each of them defines a different sequence of operation that should be performed by the system during the whole trial duration.

The **training** modality is related to a period in which some parameters of the user VCAs are extracted in order to tune the system. During this phase, one asks the subject to perform a VCA in order to collect reference data. This procedure is repeated for all the VCAs that the subject intends to use, and over many trials. Routines that automate this process are also provided.

The **testing** modality is devoted to the recognition of a VCA and to triggering an event that provides the possibility of giving feedback to the user. Usually, features extracted from the biological input signals are compared in some way with those extracted during the training sessions. This is the modality to be used once the whole system is tuned.

The **setup** modality operates between the previous two: the system asks the subject to perform one of the VCAs, and then tries to recognize it. It is used to evaluate the system’s performance as it automatically counts the number of correct and incorrect trials.

The **setup** modality is related to the hardware configuration and to the loading and saving of the references and other parameters computed during the previous or current training sessions.

The BF++ framework relies heavily on many object-oriented programming features and techniques (templates, virtual functions [9], design patterns [10]) to simplify the programming effort. It is designed to maximize efficiency. For this reason, it is not necessary that a CBF application provide all of the four mentioned modalities. This can be determined either at compile time or as run-time configuration capability. If this choice is made at compile time (this requires minimal effort), RAM and disk space can be saved, since many routines and processing capabilities are not needed for the running modality and can therefore be omitted in the final wearable application. For example, in some BCI engines (Signal Space Projection, Mahalanobis distance [11]) matrix inversion or single value decomposition routines are necessary only during the training sessions. Furthermore, although a patient performs some tests and all training in a laboratory and on a Desktop PC-based system since the collected data can be huge (e.g. 500 MB typical for an EEG based application), only a limited set of data is required in the running modality (a few KB in common BCI applications).

The BF++ framework provides a built-in file format based on the XML technology [12] that is portable across different platforms. Since the XML file structure is extensible, it can contain the acquired data, event markers that can be used for offline analysis, parameters computed during the previous training sessions, or whatever is considered necessary in the specific application domain. The XML technology also allows easy data-sharing among different laboratories. Moreover, it can be successfully used in a scenario where a laboratory and a mobile system coexist. In this case, it is then sufficient to generate, on the Desktop side, the XML file which includes all the features and the reference data that are required to operate in the running modality (extracted and computed during the training sessions) and then transfer it to the wearable device.

### III. RESULTS

The EEG signals were amplified and sampled at 128 Hz using a commercial 16-channel/12 bit portable EEG recorder (Halley by EBNeuro, [13]). The training sessions were performed on a desktop PC (Pentium II, Windows 98, 256MB RAM) while the running and testing sessions were performed on the desktop and wearable platforms. For the Compaq iPAQ device, only the behaviour was simulated. The data were transferred 8 times per second using a 115K baud serial port. Two desktop Win32-based BCI systems were ported to the Windows CE platform. They used two linear classifiers described in the literature [11], Signal Space Projection (SSP) and Mahalanobis distance, which processed EEG spectral data computed by means of FFT routines built in the BF++ framework. The FFT computation was the most power-demanding processing task of the application. With the SSP and Mahalanobis engines, it was possible to acquire and process 6 channels simultaneously on the 100MHz CPUs wearable systems, while the RLC Arm-Plus and the Compaq iPAQ (this last implementation was only simulated) were able to use all the 16 channels available from the EEG headset.

No changes to the BF++ code and to the engine implementation were required and just a recompilation of the source code was necessary for the main modules. The user interface was also easily converted from that of the Win32 system, as it mainly used features (e.g. interface controls) available under both platforms.

The efficiency of the system is also evident by two measures: 1) the SSP or the Mahalanobis engines were implemented with less than 50 lines of source code; and 2) the whole application size was less than 200 KB for all the proposed wearable configurations. This clearly suggests that the BF++ framework has the potential to be successfully employed in even more constrained situations (microcontrollers or single chip PC).

Only the acquisition layer required significant programming (about 500 lines of code) as the communication protocol was completely rewritten.
IV. DISCUSSION

BF devices currently described in literature, including those applied to the recognition of brain states, are now switching from desktop-based hardware to portable computer-based devices [14]. The ultimate goal of a practical BF device is to make it easily wearable. The BF++ framework, combined with low-cost CPU boards, provides an efficient and affordable way to achieve this target, even for a BCI application, one of the most power-demanding among BF applications. BF++’s cross-platform features also minimize the effort of migrating from one platform to another, thus maximizing reuse of resources (either data or algorithms). Finally, as it was based only on well-defined standards, it guarantees a relatively easy transition whenever new CPU boards or PocketPC models become available and needed for more powerful or smaller BF systems.

Acknowledgments

The work presented here is dedicated to the memory of Renato Grasso and Aleksander Kostov. The authors would also like to thank an anonymous referee for providing many suggestions that significantly improved the readability and value of the whole paper.

REFERENCES


