A Platform-Independent Open-Source Feedback Framework for BCI Systems

B. Venthur[1], B. Blankertz[1]

[1]Machine Learning Laboratory, Berlin Institute of Technology, Berlin, Germany

venthur@cs.tu-berlin.de

Abstract

This paper introduces the Pythonic Feedback Framework which provides a platform independent framework to develop BCI feedback applications in Python. It was designed to make the development of feedback applications as easy as possible. Existing solutions have either been implemented in C++, which makes the programming task rather tedious, especially for non-computer-scientists, or in Matlab, which is not well suited for more advanced visual (flickering is inavoidable which is unconfortable for the user and has side effects in the EEG) or auditory feedback applications.

This framework solves this problem by moving the feedback implementations to a general purpose, and easy to learn language like Python. Python provides many so called bindings to other libraries, which allow it to develop high quality multimedia feedback applications, with little effort.

The framework communicates with the rest of the BCI system via a standardized communication protocol using UDP and XML and is therefore suitable to be used with any BCI system that may be adapted to send its control signal via UDP in the specified format.

Having such a general feedback framework will also foster a vivid exchange of feedback applications between BCI groups, even if individual system for processing and classification are used.

1 Motivation

The motivation for the development of the presented feedback framework was to facilitate the implmentation of high quality BCI feedback applications and to allow the interchange of such applications between BCI groups. Accordingly, the feedback framework was designed as a standalone program which may receive the control signal from a BCI system via a standardized protocol. In order to make the implementation of new applications easy, many of the basic tasks common to most feedback applications are accomplished in the framework and have not to be re-implemented in the individual feedback applications. As programming language Python was chosen, because it is easy to learn (interpreted language) and recognized as a very mature and stable language which runs platform independent and already includes a very comprehensive class library. If Python is too slow for certain time critical methods, it is possible to write those parts in C/C++ and call them within Python. Most interesting about Python are the so called bindings. Bindings are Python libraries which make other libraries, written in other languages than Python, available to Python. We found Pygame (http://www.pygame.org/), a set of Python modules designed for writing games a very useful resource for writing feedback applications. Pygame allows to create high quality games and multimedia applications in Python. Since Python offers countless other bindings like bindings to OpenGL, it is even possible to create feedbacks with complex 3D graphics and special effects.

In contrast to compiled languages like C++, Python is also known to be very easy to learn. Users with Matlab experience will find the syntax of Python familiar.

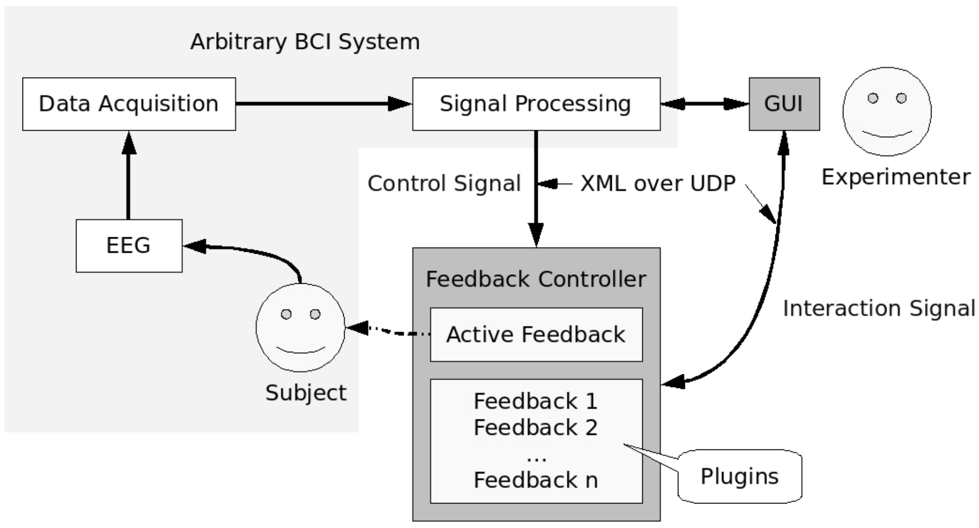

Figure 1: Setup of an BCI experiment using the Pythonic Feedback Framework. The Feedback Controller receives the control- and interaction signlas via UDP and XML.

2 Overview of the Framework

Figure 1 shows the setup of a generic BCI system using our Feedback Framework. The subject is wearing a EEG cap and sitting in front of a computer which runs a feedback application. Brain signals are collected and submitted to the data acquisition and signal processing units. The signal is processed and the result (control signal) is sent to the Feedback Controller which processes the incoming signal and forwards it to the feedback. The feedback application translates the control signal into a visual, audible and/or tactile output.

The experimenter can remote-control the feedback and manipulate its variables via the GUI, which also sends singals to the Feedback Controller. Those signals are called interaction signals.

The framework communicates with the rest of the BCI system via a standardized communication protocol using User Datagram Protocol (UDP) and Extensible Markup Language (XML) and is therefore not bound to a single BCI system, but should be usable with any BCI system providing control signals of some kind.

The feedbacks are realized as plugins of the Feedback Controller.

3 Components of the Framework

The feedback controller (Sec. 3.1) and the feedback base classes (Sec. 3.2) are components that are general and take over the much of the programming load for the implementation of new feedback applications by providing much of the general functionality.

3.1 Feedback Controller

The Feedback Controller manages the communication between the feedback and the rest of the BCI system. It acts like a server, collecting control- and interaction signals over the network. Once the Feedback Controller ist started, it is fully remotely controllable from the GUI: it is possible to load feedbacks, manipulate their variables, start, pause and stop them. The data is sent in XML format and must be therefore translated into Python compatible data structures and commands. The Feedback Controller takes care of this. It sets feedback variables if necessary and calls the appropriate event methods of the feedback.

Feedbacks are realized as plugins of the the Feedback Controller. Since the Feedback Controller takes care about the whole communication between the feedback and the rest of the BCI

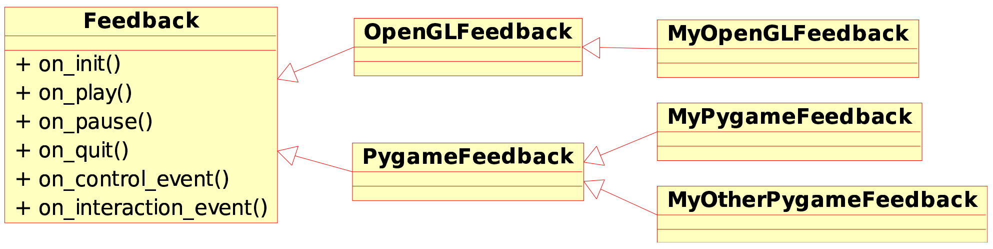

Figure 2: Possible classdiagram: OpenGLFeedback and PygameFeedback are derived from the Feedback base class and implement common functionality needed in all OpenGL- and Pygame feedbacks.

system, implementing feedbacks is straight forward. Furthermore, since there is a uniform interface between the feedback and the Feedback Controller, it is even possible, to exchange feedbacks between various BCI systems of different BCI groups when using this framework.

3.2 Feedback Base Class

The Feedback base class is the interface to the Feedback Controller's plugin system. The base class provides methods, the Feedback Controller needs to communicate with the feedback. By subclassing the Feedback base class, the derived class becomes a valid and ready-to-use feedback, available to the Feedback Controller.

Feedbacks are event driven. Whenever the Feedback Controller receives a signal, it calls the appropriate methods of the feedback to notify it: an incoming control signal causes an `on_control_event` on the feedback, an incoming interaction signal with the play command an `on_play`, and so on. To react on such events, only the respective `on_`-method of the feedback has to be implemented. It is not necessary to implement all available events of the Feedback base class, if the Feedback Controller triggers an event which was not implemented in the feedback, nothing happens.

The object oriented approach of this framework makes it possible to further simplify the development of feedbacks: If it becomes obvious that for example feedbacks using Pygame often share the same blocks of code, it is possible to derive a PygameFeedback baseclass from the Feedback base class, which already implements the shared functionality. Actual feedbacks using Pygame can be derived from the PygameFeedback baseclass which can reduce the amount of code per feedback drastically. Figure 2 illustrates the example.

3.3 GUI

The GUI is responsible for sending interaction signals to the Feedback Controller. It acts like a remote control for the Feedback Controller and the running feedback: it allows to load, un-load feedbacks, modify their variables, start, stop and pause them.

Once the GUI ist connected to the Feedback Controller, the Feedback Controller publishes all available feedbacks to the GUI. The experimenter can now select the desired feedback in the GUI and tell the Feedback Controller to load it. Once the Feedback Controller has loaded the feedback, the feedback's variables are automaticly published to the GUI. The GUI presents those variables and their values in tabular form, allowing to modify their values or create new ones and send them back to the Feedback Controller where they are directly applied to the running feedback.

The GUI is written in Python and QT and runs—like the rest of the framework—platform independently.

3.4 Documentation and Examples

Part of the framework is also a complete documentation of the system and its interfaces. The documentation also provides a guide how to write own feedback applications using this framework. Several well documented examples, explaining every major aspect of the feedback implementation, are included.

4 A simple feedback example

The following listing shows a trivial feedback written with the framework. Although it does nothing but printing the current control signal two times per second, it already shows the basic structure of every feedback.

```
from Feedback import Feedback
import time

class ExampleFeedback(Feedback):

    def on_init(self):
        print "Feedback successfully loaded."
        self.quitting, self.quit, self.pause = False, False, False

    def on_quit(self):
        self.quitting = True
        print "Waiting for main loop to quit."
        while not self.quit:
            pass

    def on_play(self):
        self.quitting, self.quit = False, False
        self.main_loop()

    def on_pause(self):
        self.pause = not self.pause

    def main_loop(self):
        while 1:
            time.sleep(0.5)
            if self.pause:
                continue
            elif self.quitting:
                break
            print self._data
        print "Left main loop."
        self.quit = True
```

There are three variables which control the behavior of the feedback: `pause` tells the feedback to pause it's action, `quitting` tells the feedback to quit it's main loop and `quit` is set when the main loop has quit.

The heard of the feedback is the `main_loop` method. As the name suggests, it contains an infinite loop where each iteration represents a tick (a very short amount of time) of the running feedback. Each tick, the feedback checks the aforementioned variables `pause` and `quitting` and decides what to do. If `pause` is set, the feedback just skips this tick, if `quitting` is set, it leaves the loop and sets the `quit` method. If none of the two variables is set, it just executes the tick, in this case by printing the content of the control signal.

The event `on_play` starts the main loop, `on_pause` and `on_quit` control the main loop's behavior by setting the `pause` and `quitting` variables. `on_quit` does a bit more than just setting the `quitting` variable: after the variable has been set, it waits until the main loop has quit by repeatingly checking the `quit` variable. Only after `quit` has been set, the `on_quit` method returns, which tells the Feedback Controller that the feedback has successfully terminated.

5 Communication with the rest of the BCI system

In order to couple the Framework as loosely as possible with the rest of the BCI system and thus allowing to support many different BCI systems, the Feedback Controller receives the control- and interaction signals via User Datagram Protocol (UDP), a very lightweight network protocol supported by all major operating systems and programming languages. UDP servers and -clients are trivial to implement and allow the communication between programs on different machines in the network or on the same machine.

The the content of the signals is send via Extensible Markup Language (XML) which is a well known standard for exchanging information especially over internet. Libraries to parse XML files or creating new ones are available for most modern programming languages.

The utilization of UDP and XML as a standardized interface to communicate with the framework, should make it very easy to adapt most existing BCI systems to use this framework to develop Feedbacks in Python.

6 Requirements

The framework itself has very few dependencies. It needs Python 2.4 or higher and optionally pyParallel (`http://pyserial.sourceforge.net`) to utilize the parallel port to send markers to the EEG acquisition system. No requirements are imposed regarding the operating system. The framework will run on every platform which is supported by Python.

To use the GUI which uses Python's QT bindings, PyQT (`http://www.riverbankcomputing.com/software/pyqt/`) is needed. PyQT also runs on every major platform.

To use this framework in a specific BCI system, the BCI system should be able to send the control signal via XML and UDP. Modifying existing BCI systems to create valid XML files containing the control signal and sending them over UDP, should be fairly easy regardless of the programming language being used.

7 Conclusion

The Pythonic Feedback Framework provides a solution for writing high quality BCI feedback applications with minimal effort. Through the use of a standardized interface using a standard protocol for the communication with the BCI system, this framework is very generic and it should be easily adaptable to most existing BCI systems.

Moreover such a unified feedback framework creates the unique opportunity of exchanging BCI feedback applications between BCI groups, even if individual systems are used for processing and classification.

The presented framework runs on every major platform (Linux, Mac and Windows), is Free Software and licensed under the terms of the GNU General Public License (GPL) for non-commercial purposes.