The Pyff Lecture

or: How I Learned to Stop Worrying and Love the Python

Bastian Venthur

Berlin Institute of Technology

2012-09

Outline

Outline

Who am I?

- PhD student at the Berlin Institute of Technology
- Studied Computer Science at the Free University of Berlin
- Created Pyff as my Diploma Thesis

Overview: BCI System

Overview: BCI System

Motivation

Why Pyff?

- Before Pyff everything was written in matlab
- Matlab is not a general purpose programming language
- Matlab is not well suited for multi media

Pyff

Framework for Feedback and Stimulus Presentation written in Python. Allows you to write your own Feedback and Stimulus applications with minimal effort

Features

- BCI system independent
- Written in Python
- Free- and Open-Source Software
- Comes with many ready to use standard paradigms
- Comes with many templates for paradigms and experiments

Why Python?

- Free- and Open-Source Software
- Established and well-known
- General purpose programming language
- Supports may programming paradigms (imperative, OOP, functional)
- Awesome standard library (batteries included)
- Smooth learning curve
- Matplotlib, Numpy, Scipy

! Excellent alternative to Matlab

Outline

Pyff's Components

1. Feedback Controller
2. GUI
3. Set of Feedback Base Classes
4. Set of ready-to-use Feedbacks
5. XML

Data Flow

Data Flow cont'd

The Feedback Controller consumes two different kinds of signals:

Control Signal

Processed or raw EEG data

Interaction Signal

Configuration data et. al.

The Feedback Controller

- The “main program”
- Starts the GUI and waits for incoming commands

Behind the scenes

- Opens a server port and waits for incoming control/interaction signals
- Initializes, Starts, Stops, etc. Pyff Applications
- Forwards incoming Control Signals to the Pyff Application

The GUI

Pyff's Graphical User Interface

- Browse available Feedbacks
- Initialize, start, stop, etc., Feedbacks
- Inspect and modify instance variables

Feedback Base Classes

At the top of the class hierarchy is Feedback.py. It defines a set of methods the Feedback Controller relies on to communicate with every Feedback.

- Derived classes for special purposes are provided
- I.e. the Pygame Feedback base class inherits a main loop and implements a lot of code every Pygame Feedback needs

Feedback Base Class

The on Events

Feedbacks are event-driven, your Feedback runs and some of its methods get called by the Feedback Controller:

foo bar

XML Protocol

You don't deal with it directly all serialization is done by Pyff.

- Serializes Python's basic data types (int, float, str, lists, dicts, sets, etc.)
- Preserves variable names, type and value
- Allows for interoperability
  - Allows to send matlab variables to Python (and back)
- Loosely couples Pyff with the rest of the BCI system

But

- A bit overkill
- Will probably be replaced with JSON

Pyff's Components

Does that slide make more sense now?

1. Feedback Controller
2. GUI
3. Set of Feedback Base Classes
4. Set of ready-to-use Feedbacks
5. XML

Outline

Starting Pyff

Starting Pyff...

On Windows

```
python FeedbackController.py
```

Everywhere Else

```
./FeedbackController.py
```

... will start the Feedback Controller and the GUI

- The FC waits for incoming data
- The GUI waits for user input

The Feedback Controller has several options

For an overwiew:

```
./FeedbackController.py --help
```

Using the GUI

Overview

Using the GUI cont'd

Starting a Feedback et al

1. Select Feedback
2. Initialize Feedback
3. Start Feedback
4. Pause Feedback
5. Stop Feedback
6. Quit Feedback
7. (Connect to Feedback Controller)

Using the GUI cont'd

Dealing with the Feedback's variables

Demo

Outline

Checklist for a New Feedback

1. Write your Feedback and derive it from Feedback or one of its base classes

```
$ cat /tmp/foo/foo.py
from FeedbackBase.Feedback import Feedback

class Foo(Feedback):

    def on_init(self):
        print 'foo'
```

2. Write a feedbacks.list

```
$ cat /tmp/foo/feedbacks.list
foo.Foo
```

3. Start the Feedback Controller with the -a parameter pointing to the directory containing the feedbacks.list file

```
./FeedbackController.py -a /tmp/foo/
```

A Trivial Feedback

```
import time

from FeedbackBase.Feedback import Feedback


class MyFirstFeedback(Feedback):

    def on_init(self):
        self.logger.debug("Feedback successfully loaded.")

    def on_quit(self):
        self.logger.debug("Feedback quit.")

    def on_play(self):
        self.logger.debug("Play.")
        self.send_parallel(0x1)
        self.running = True
        while self.running:
            print self._data
            time.sleep(0.1)

    def on_stop(self):
        self.logger.debug("Stop.")
        self.send_parallel(0x2)
        self.running = False
```

feedbacks.list

Purpose

Tell the Feedback Controller where to look for Feedback classes.

Syntax

- Plain text file, one entry per line (usually only one)
- Import path to the class relative to the location of the feedbacks.list file

Example

```
$ cat /tmp/bar/foo.py
from FeedbackBase.Feedback import Feedback

class Foo(Feedback):
    # Foo's code...

$ cat /tmp/bar/feedbacks.list
foo.Foo
```

Ressources

Pyff Homepage	`http://bbci.de/pyff`
git Repository	`http://github.com/venthur/pyff`

Questions?

Fin