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

Bastian Venthur

Berlin Institute of Technology

2012-09

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

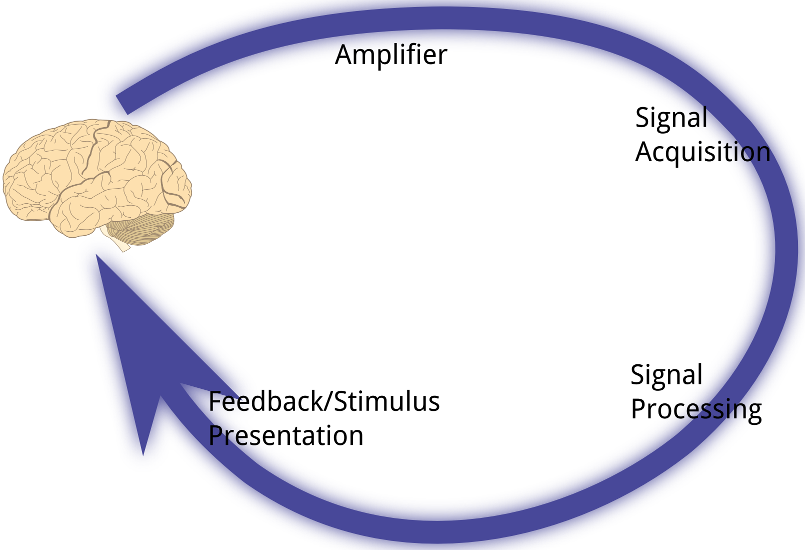
Contact me!

- bastian.venthur@tu-berlin.de
- @bastianventhur

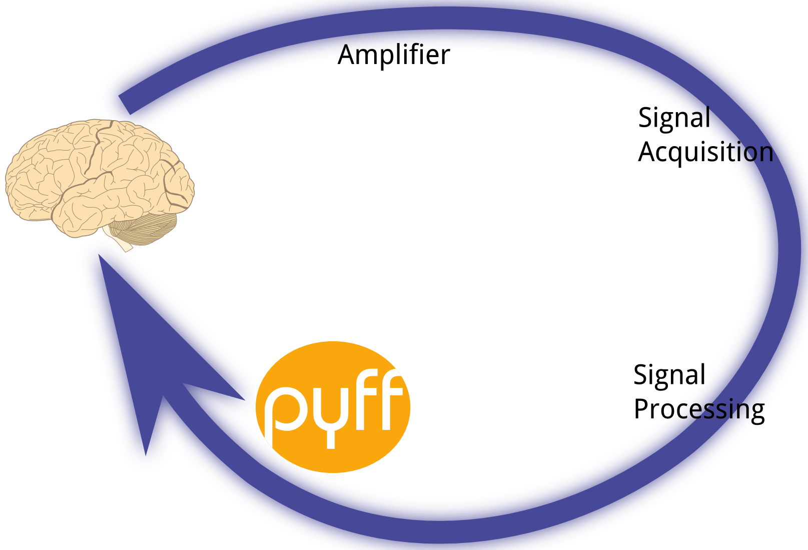
Outline

Outline

Overview: BCI System

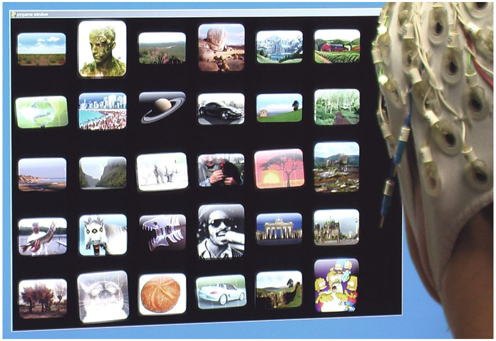


Overview: BCI System



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

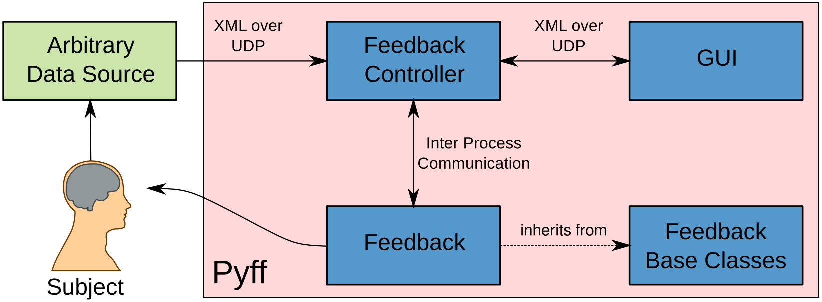
Why Python?

- 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

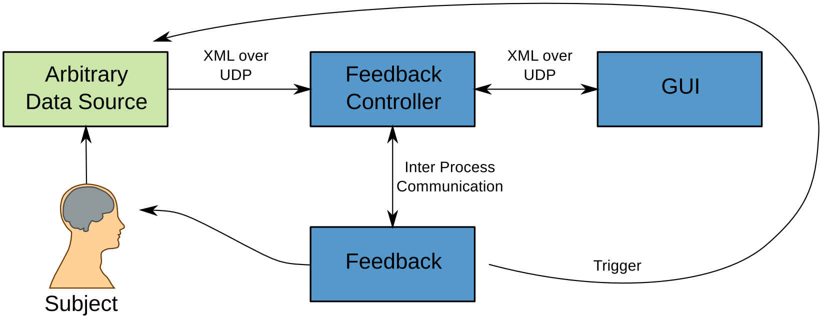
Outline

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


Arbitrary Data Source
XML over UDP
Feedback Controller
XML over UDP
GUI
Inter Process Communication
Feedback
Trigger
Subject

Data Flow cont'd

There's two different kinds of data

The Feedback Controller consumes two different kinds of signals:

Control Signal	Interaction Signal
Processed or raw EEG data	Configuration data et. al.

The Feedback Controller

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

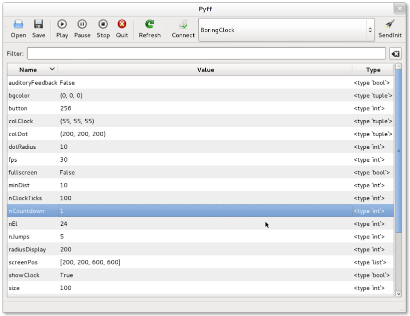
Behind the scenes

- 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

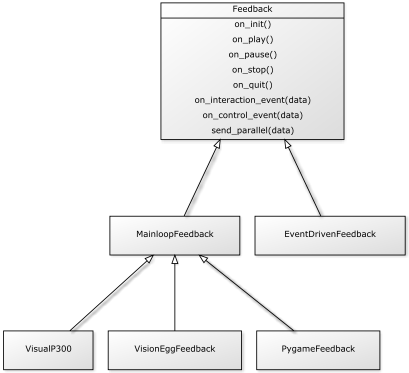


Name	Value	Type
auditoryFeedback	False	<type 'bool'>
bgcolor	(0, 0, 0)	<type 'tuple'>
button	256	<type 'int'>
colClock	(55, 55, 55)	<type 'tuple'>
colDot	(200, 200, 200)	<type 'tuple'>
dotRadius	10	<type 'int'>
fps	30	<type 'int'>
fullscreen	False	<type 'bool'>
minDist	10	<type 'int'>
nClockTicks	100	<type 'int'>
nCountdown	1	<type 'int'>
nEl	24	<type 'int'>
nJumps	5	<type 'int'>
radiusDisplay	200	<type 'int'>
screenPos	[200, 200, 600, 600]	<type 'list'>
showClock	True	<type 'bool'>
size	100	<type 'int'>

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:

on_init	Feedback gets initialized
on_play	Feedback gets started
on_pause	Feedback gets paused
on_stop	Feedback gets stopped
on_quit	Feedback gets terminated
on_control_event	Feedback got data from the EEG
on_interaction_event	Feedback got config data

Feedbacks run in a different Process!

XML Protocol

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

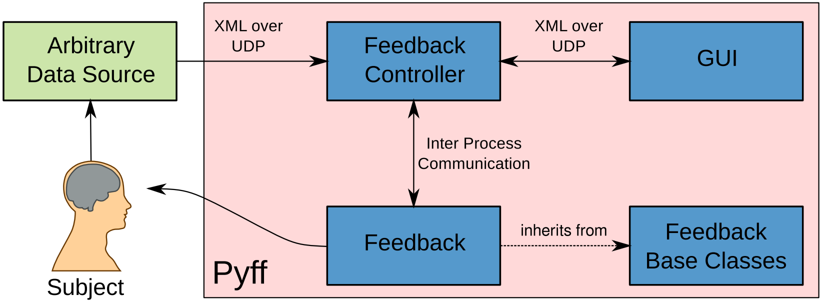
- Serializes Python's basic data types (int, float, str, lists, dicts, sets, etc.)
- Preserves variable names, type and value
- Allows for interoperability
  - Allows to send matlab variables to Python (and back)
- Loosely couples Pyff with the rest of the BCI system

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

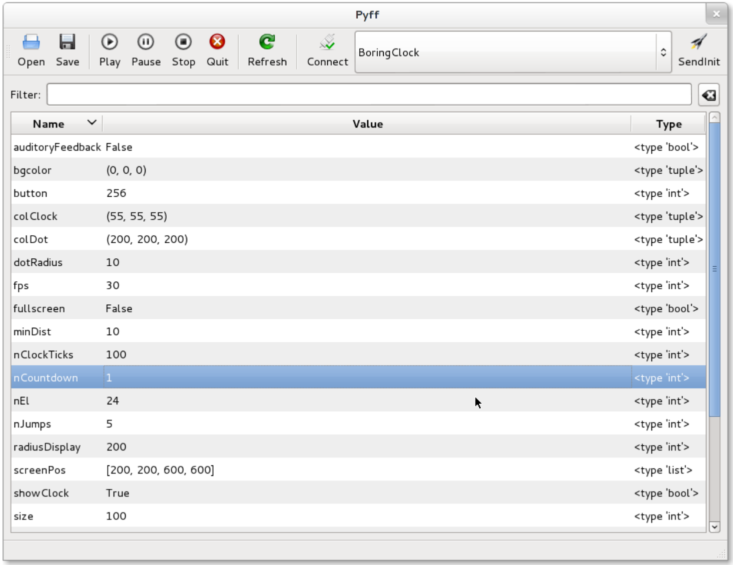
- The FC waits for incoming data
- The GUI waits for user input

The Feedback Controller has several options

For an overwiew:

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

Using the GUI
Overview

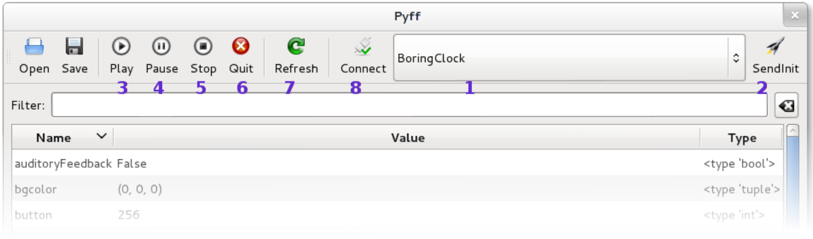


BoringClock

Name	Value	Type
auditoryFeedback	False	<type 'bool'>
bgcolor	(0, 0, 0)	<type 'tuple'>
button	256	<type 'int'>
colClock	(55, 55, 55)	<type 'tuple'>
colDot	(200, 200, 200)	<type 'tuple'>
dotRadius	10	<type 'int'>
fps	30	<type 'int'>
fullscreen	False	<type 'bool'>
minDist	10	<type 'int'>
nClockTicks	100	<type 'int'>
nCountdown	1	<type 'int'>
nEl	24	<type 'int'>
nJumps	5	<type 'int'>
radiusDisplay	200	<type 'int'>
screenPos	[200, 200, 600, 600]	<type 'list'>
showClock	True	<type 'bool'>
size	100	<type 'int'>

Using the GUI cont'd

Starting a Feedback et al



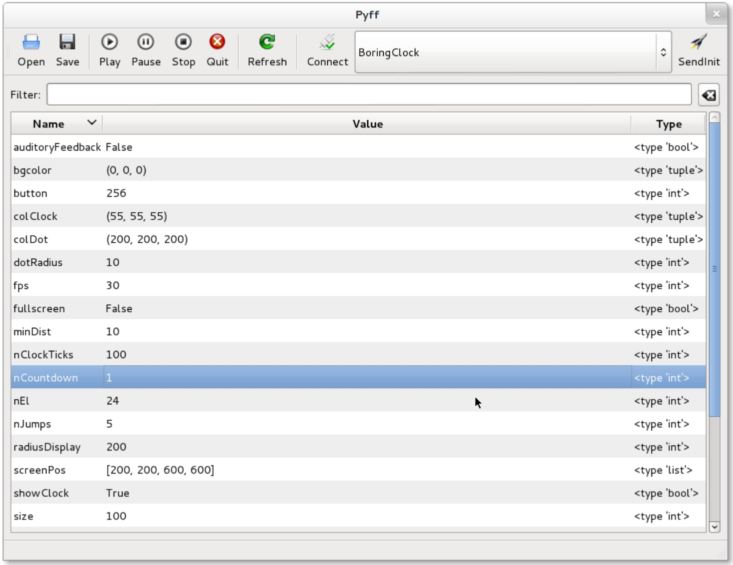
BoringClock

Name	Value	Type
auditoryFeedback	False	<type 'bool'>
bgcolor	(0, 0, 0)	<type 'tuple'>
button	256	<type 'int'>

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



Name	Value	Type
auditoryFeedback	False	<type 'bool'>
bgcolor	(0, 0, 0)	<type 'tuple'>
button	256	<type 'int'>
colClock	(55, 55, 55)	<type 'tuple'>
colDot	(200, 200, 200)	<type 'tuple'>
dotRadius	10	<type 'int'>
fps	30	<type 'int'>
fullscreen	False	<type 'bool'>
minDist	10	<type 'int'>
nClockTicks	100	<type 'int'>
nCountdown	1	<type 'int'>
nEl	24	<type 'int'>
nJumps	5	<type 'int'>
radiusDisplay	200	<type 'int'>
screenPos	[200, 200, 600, 600]	<type 'list'>
showClock	True	<type 'bool'>
size	100	<type 'int'>

Demo

Outline

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
```

Anatomy of Game Like Applications

```
while True:     # <- Main Loop
    Get Keyboard, Mouse, etc. inputs.
    Compute next step(s)
    Redraw Screen
```

The inner part of the loop we call a tick.

Anatomy of Game Like Applications

```
while True:      # <- Main Loop
    Get Keyboard, Mouse, etc. inputs.
    Compute next step(s)
    Redraw Screen
```

The inner part of the loop we call a tick.

```
while True:      # <- Main Loop
    if paused:
        pause_tick()
    else:
        play_tick()

def play_tick():
    Get Keyboard, Mouse, etc. inputs.
    Compute next step(s)
    Redraw Screen

def pause_tick():
    ...
```

Pygame Feedback Base Class

Important methods and variables

Methods

Inherited from Mainloop Feedback base class

- play_tick
- pause_tick

Pygame Feedback Base Class

Important methods and variables

Methods

Inherited from Mainloop Feedback base class

- play_tick
- pause_tick

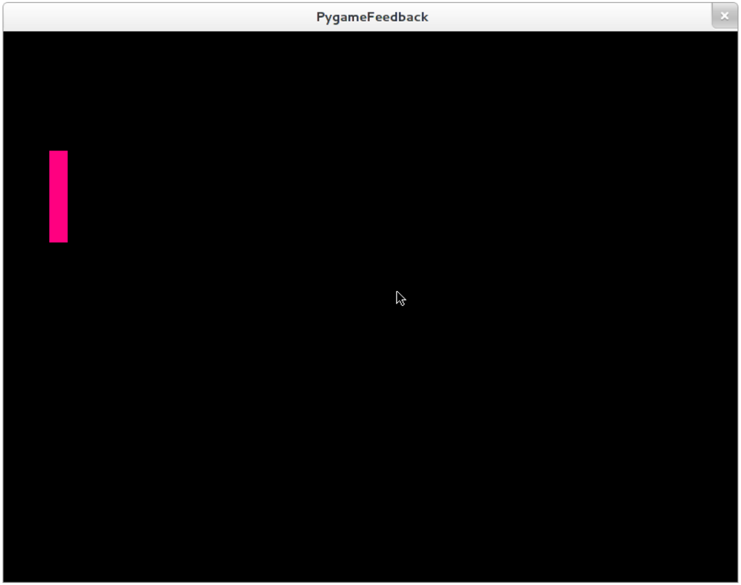
Variables

FPS	Frames per second
screenSize	Width and height of the window
elapsed	Time since last tick (in seconds)
lastkey_unicode	Last key pressed

... and many more

Enter Pong!

Well...



Pong
Classy!

```
import pygame

from FeedbackBase.PygameFeedback import PygameFeedback


class Pong(PygameFeedback):

    def init(self):
        PygameFeedback.init(self)
        # color, pos, width and height of the paddle
        self.color = [255, 0, 128]
        self.pos = 100
        self.width = 20
        self.height = 100

    def on_control_event(self, data):
        d = data['cl_output']
        if d > 0: self.pos += 5
        elif d < 0: self.pos -= 5
```

Pong

```
def play_tick(self):
    # clear the background
    self.screen.fill(self.backgroundColor)
    # check for keyboard input
    if self.keypressed:
        if self.lastkey_unicode.lower() == 'w':
            self.pos -= 5
        elif self.lastkey_unicode.lower() == 's':
            self.pos += 5
        self.keypressed = False
    # check if paddle out of screen
    if self.pos < 0: self.pos = 0
    if self.pos > self.screenSize[1]: self.pos = self.screenSize[1
    # draw the paddle
    pygame.draw.rect(self.screen, self.color,
                     [50, self.pos - self.height/2,
                      self.width, self.height])
    pygame.display.flip()
```

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'
```

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
```

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/
```

Ressources

Pyff Homepage	`http://bbci.de/pyff`
git Repository	`http://github.com/venthur/pyff`
Exercises and Material	`http://github.com/venthur/pyff-lecture`

Questions?

Fin