Research Paper Documentation

Classification of Mental States using EEG

Applied Computer Science at the Baden-Wuerttemberg Cooperative State University Stuttgart

from

Stefan Heßler, Jan Forster, Lucas Bader

Advisor: Prof. Dr.-Ing. David Suendermann-Oeft

Table of Contents

1. Introduction

This documentation describes the software used to run experiments on different EEG devices (i.e. Emotiv EPOC headset[1] & NeuroSky MindWave[2]) as presented in the research paper "Classification of Mental States using EEG". The documentation serves the purpose to enable the reader to repeat the experiments and to conduct own research based on the software shown. It should not be viewed as a full code documentation but it explains all critical elements of the software.

First the different software components are introduced. Afterwards, the components are explained in-depth starting with the interface components that realize the connection between the PC and the EEG headset devices. The last section explains how the software can be used to start the respective experiments that are discussed in the corresponding research paper.

Please note that this documentation only describes the software used during the research and has to be seen as an addition to the research paper. For the interpretation of the data, experimental procedures and research results, please consult the paper. It is advised to read the research paper beforehand to understand the experimental software.

2. Software Components

2.1 Overview

Overall, the software is divided by headset interface and specific experiment. Firstly there is a component to run the eye state capture experiment and secondly the Binarual Beats experiment. Both components build on the interface components for the MindWave and the EPOC headset. These interface components are able to get the raw EEG data sets from the according headset devices and can therefore easily be interchanged. Both experiment components have additional experiment specific code (e.g video capture and face recognition). Most of the code is implemented in the Python language. On the lowest level, bash scripts have been implemented that run all appropriate software components for a specific experiment. Those will be explained in the *Usage* section of this documentation.

This section will first explain the interface components for the different headset devices. Afterwards the code for both experiment types and their additional functionality will be discussed. This includes functionality for capturing the eye state based on the OpenCV[3] face recognition as well as the software for playing back different binaural beats while providing visual stimuli as explained in the research paper.

1 https://www.emotiv.com/epoc/
2 http://store.neurosky.com/products/mindwave-1
3 http://opencv.org/

2.2 Emotiv EPOC Headset Interface

To realize the connection to the Emotiv EPOC EEG headset the open-source library python-emotiv[4] by Ozan Çağlayan, from the Computer Engineering Department of Galatasaray University is used. The developer of said library reverse-engineered the protocol that is used by the headset to acquire the raw EEG data on Linux.

As the library is not documented very well, a pragmatic approach was used to incorporate the headset interface within the experimental software. Instead of using the raw library functions to get single data sets from the EPOC headset, example code was discovered within the library that records data over a predefined time period. Since the experiments are well-defined, it is possible to run the python-emotiv software simultaneous to the experiment code. As mentioned before, this is implemented in an experiment and headset specific bash script.

The following code snippet shows how this can be used. The record-data.py example is run with exactly one parameter that specifies the time period for the capture in milliseconds. The software captures the EEG data and saves the raw values of each sensor to a matlab file such as “emotiv-08-04-2014_14-49-53.mat”.

```
./python-emotiv/examples/record-data.py 770
```

Listing 1: Emotiv EPOC interface call

2.3 NeuroSky Mindwave Headset Interface

The interface for the NeuroSky MindWave device is implemented in the files *mindwave.py* and *eegCapture.py.* The *mindwave.py* component carries out the serial connection to the headset via the appropriate USB port. The central class is the *Headset* class. It implements a *DongleListener* that runs in an endless loop and captures the data coming from the headset. For each package sent by the headset (sampling rate 512 Hz) the EEG data is saved in the headset property of the *Headset* object as follows.

```
self.headset.raw_data # raw EEG value in mV.

# multi-band EEG data
self.headset.delta
self.headset.theta
self.headset.low_alpha
self.headset.high_alpha
self.headset.low_beta
self.headset.high_beta
self.headset.low_gamma
self.headset.mid_gamma
```

Listing 2: EEG data values for the Mindwave headset

4 https://github.com/ozancaglayan/python-emotiv

Within the *eegCapture.py* file the *RawDataPlotting* class is implemented that uses the Headset class to get a discrete data set from the MindWave device. This is realized within the *getData* method of the *RawDataPlotting* class. The constructor *__init__* method uses the *Headset* class to open the connection to the headset and waits for the *headset.status* property to change to 'connected' which indicates a successful connection.

The *getData* method returns a string with the raw EEG data as well as the multi-band EEG data in a semicolon-separated fashion.

```
raw_data;delta;theta;low_alpha;high_alpha;low_beta;high_beta;low_gamma;mid_gamma
```

Listing 3: comma-separated file format

The following code shows the minimal required actions to open a connection the the MindWave device and capture a single EEG data set to a string variable.

```
EEG = eegCapture.RawDataPlotting("folder")
string = EEG.getData()
EEG.close()
```

Listing 4: Minimal action to open MindWave device and capture

2.4 Eye State Classification Experiment

The classification experiment is designed to measure the performance of each EEG headset device to detect the eye state of a subject. In order to have a performance indicator the eye state has to be tracked precisely. This process is automated. The code for the face and eye state recognition is based on the OpenCV[5] framework.

Central to the eye detection is the *EyeCapture* class within the *eyeCapture.py* source file.

5 http://opencv.org/

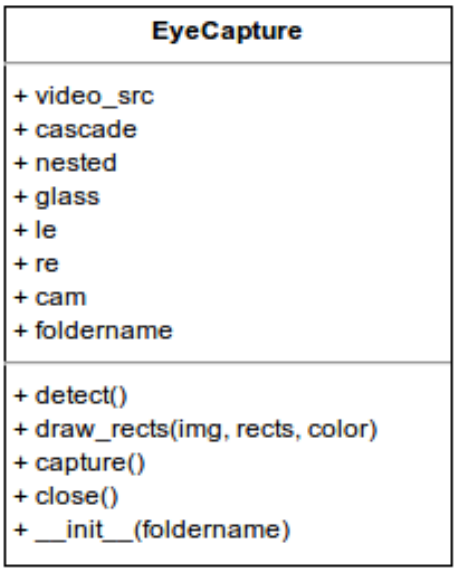

Fig 1: The EyeCapture Class

The eye detection is based on the CV2 Cascade Classifiers[6] which is essentially a cascade of trained classifiers that can be applied to a region of interest. Within the *__init__* constructor method these classifiers are loaded. Additionally, as each frame is captured as a *.jpg* image file, an output folder is created to save each frame.

```
cascade_fn = "haarcascades/haarcascade_frontalface_default.xml"
nested_fn = "haarcascade_eye.xml"
nested_glass = "haarcascades/haarcascade_eye_tree_eyeglasses.xml"
nested_le = "haarcascades/haarcascade_lefteye_2splits.xml"
nested_re = "haarcascades/haarcascade_righteye_2splits.xml"

os.mkdir(self.foldername + "/eyeCaptureImages")

print "eyeCapture: setting CascadeClassifiers"
self.cascade = cv2.CascadeClassifier(cascade_fn)
self.nested = cv2.CascadeClassifier(nested_fn)
self.glass = cv2.CascadeClassifier(nested_glass)
self.le = cv2.CascadeClassifier(nested_le)
self.re = cv2.CascadeClassifier(nested_re)

self.cam = create_capture(0)
```

Listing 5: OpenCV2 Cascade Classifier for eye recognition

6 http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

The cascade is built up by a facial classifier, an eye classifier (with a classification for glasses), and 2 classifiers for each eye.

The *detect* method uses the cascade classifier to identity the eye state (open, close). The approach is pragmatic. If eyes are detected, they have to be open. If no eyes are detected, but a face is detected, they are closed.

The *capture* method essentially returns a string with the detection result. It includes a timestamp of the detection which is combined with the classification value separated by a semicolon to be compatible with the overall output format of the software. If the eye state is open, a value of 0 is returned, else a value of 1 is returned. If no face was detected, a value of *n.e* is returned.

```
subrects_fn = self.detect(roi.copy(), self.nested)
subrects_glass = self.detect(roi.copy(), self.glass)
subrects_le = self.detect(roi.copy(), self.le)
subrects_re = self.detect(roi.copy(), self.re)
...
if not len(subrects_fn) == 0:
        string = string + "1;"
elif not len(subrects_glass) == 0:
        string = string + "1;"
elif (not len(subrects_le) == 0) or (not len(subrects_re) == 0):
        string = string + "0;"
else:
        string = string + "n.e.;"
```

Listing 6: Eye recognition for eye state classification

The components for the eye state detection and EEG capture are combined in the *captureEEGEye.py* component which is an executable Python file. The usage will be explained in the last section of this documentation.

The component is responsible for initializing the headset connection and the eye capture. All files will be saved to a given directory (either specified by the user or to a default folder). Note that the headset initialization is only required for the MindWave headset device. The EPOC headset code is run from a bash script in parallel and is mapped by timestamp later on. The *csv* file is initialized with the column headers.

```
of = open(foldername + "/values.csv", "w")
of.write("time_start;time_eye;eyestate;time_eeg;raw_eeg\n")
print "initializing EEG..."
rawEEG = eegCapture.RawDataPlotting(foldername)
print "initializing camera..."
camera = eyeCapture.EyeCapture(foldername)
```

Listing 7: Initialization of the captureEEGEye.py component

After initializing all necessary components, the capture is started and can be stopped by the escape button on the keyboard. Within each iteration of the main loop a string is written to the output file with a timestamp, the EEG data and the camera capture data.

```
while True:
        if 0xFF & cv2.waitKey(5) == 27:
                break
        string = str(int(round(time.time() * 1000)))
                + ";" + rawEEG.getData() + camera.capture() + "\n"
        of.write(string)
        string = ""
```

Listing 8: Main Loop

In the final section of this documentation the usage of the described software component will be discussed.

2.5 Binaural Beats Experiment Component

This experiment component consists of two sub-components, one implementing the user interface and the other the binaural beats playback and EEG connection. As explained in the corresponding research paper, the subject is presented a screen with moving shapes. His task is to count the shapes. This component resides in the *shapecount.py* component and is implemented with the *pygame* Python library [7].

User Interface / Game sub-component

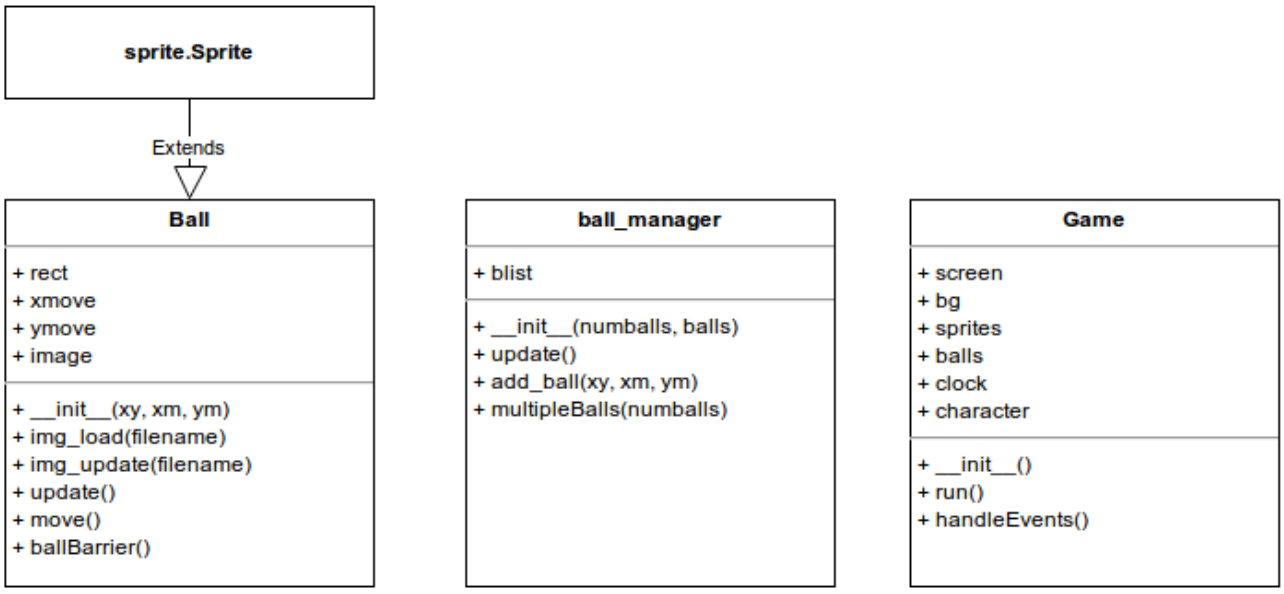

Fig 2: The shapecount.py game component

As shown in Fig. 2, the user interface component consists of three classes. The *Ball* class represents a single moving shape on the screen and extends the *sprite.Sprite* class within the *pygame* library. As such it needs to implement an *update* method, which is called for each frame and updates the movement of the shape and its barrier for collision detection.

The shape is defined by a rectangle (*rect* property), an *image* property which holds the shape's image and movement properties (*xmove* and *ymove*) that indicate the movement speed of the shape along both axes.

The *img_load* and *img_update* methods load the specified image file and initialize the surrounding rectangle and are straight-forward. The *move* method is also easy to understand as it simply adds the value in the movement properties to the current absolute positioning properties of the rectangle (*rect.x* & *rect.y*).

7 http://www.pygame.org/news.html

More interesting is the *ballBarrier* method of the *Ball* class. It checks to make sure the shape is within the bounds of the screen and carries out multiple operations if that is not the case. Firstly, the screen dimensions and the shape size are held within the global constants *SCREEN_WIDTH, SCREEN_HEIGHT* and *BALL_SIZE.*

```
if self.rect.right > (SCREEN_WIDTH + BALL_SIZE):
        self.xmove = random.randint(-2, -1)
        self.img_update('img/shape' + str(rnd_nr) + '.png')
```

Listing 9: Collision detection for the Ball class

The listing above shows one example case of the four cases that need to be handled by the barrier code. The code is executed if the shape has moved out of the right side of the screen. As described in the experiment description in the research paper, the shape is intended to randomly change its color. This is simply realized by loading a new image with a random color number as the shape files should be numbered. Additionally the *xmove* property is updated with a random number to change the speed of the shape as it is moved back to the left.

The *ball_manager* class is responsible for adding new shapes and to randomly initialize multiple shapes. It holds all its shapes in the *blist* list property. Since the implementation is trivial, it is not discussed further at this point.

The *Game* class combines the above and is responsible for bringing the shapes to life. It initializes the screen and to fill it with a background color, to initialize a clock, to set a window, etc. Core of the implementation are the *run* method which implements the main game loop (i.e. updating the sprites and drawing everything for each frame) and the *handleEvents* method that handles incoming key strokes from the user. Both methods are typical for a *pygame* implementation. Again, the implementation of both methods is straight-forward and is not discussed further.

Lastly, the *main* method within the *shapecount.py* component runs the user interface/game code.

```
def main():
   game = Game()
   game.run()
   quit()
```

Listing 10: shapecount.py main method

Binaural Beats / EEG Capture sub-component

The second sub-component serves the purpose of playing back the different binaural beats sound files and of the initialization of the EEG headset (i.e. MindWave). It also saves the captures to a *csv* file as seen in the eye state classification experiment component. Again, *pygame* is used, this time to play back audio files.

It is implemented in a procedural fashion. First all components are initialized. This includes *pygame,* the EEG device, a *pygame.mixer.Sound* object to handle the audio playback and the appropriate *csv* output file similar to the previous experiment. The following listing shows the crucial initialization tasks.

```
stages = ['noise_start', 'delta', 'noise', 'theta', 'noise', 'alpha', 'noise', 'beta', 'noise', 'gamma']
...
of = open(foldername + "/values.csv", "w")
of.write("time_eeg;stage;raw_eeg;delta;theta;low_alpha;high_alpha;low_beta;high_beta;low_gamma;mid_gamma;\n")
...
rawEEG = eegCapture.RawDataPlotting(foldername)
...
noise_sound = pygame.mixer.Sound("data/rain2.ogg")
channelA = pygame.mixer.Channel(1)
channelB = pygame.mixer.Channel(2)
channelA.play(noise_sound)
```

Listing 11: Binaural Beats initialization code

As shown above, the mixer has two channels. This is due to the fact that multiple audio files will be played back in parallel (rain background noise and binaural beats).

After all initialization tasks are done, the main loop is run which iterates over each defined stage. Crucial to this loop is the timing functionality which makes the code move on to the next stage.

```
current_time = (datetime.datetime.now() - datetime.datetime.combine(datetime.datetime.now().date(), datetime.time(0))).seconds #reset timer to current seconds
timer_sec = 0
```

Listing 12: Timer initialization

The listing above shows the timer code that starts each iteration of the loop. First, the *current_time* is set to the current time in seconds and the *timer_sec* is reset to 0 counting the duration of the current stage.

Afterwards, the stage is identified and the according sound file is played back. The implementation can be reduced to the following for each stage:

```
binaural_sound = pygame.mixer.Sound("data/STAGE.ogg")
channelB.play(binaural_sound)
```

Listing 13: Binaural Beat audio playback

Now, it is differentiated between “noise” stages and “binaural” stages. This is required by the experiment description which defines different time spans for each respective type of stage. However, the timing essentially follows the same implementation. Additional to the timing, the EEG data is captured at this point, too.

```
while timer_sec <= 120:
        string = str(int(round(time.time() * 1000))) + ";" + stage + ";" + rawEEG.getData() + "\n"
        of.write(string)
        string = ""
        timer_sec = (datetime.datetime.now()
            - datetime.datetime.combine(datetime.datetime.now().date(),
                datetime.time(0))).seconds
            - current_time
        time.sleep(0.001)
```

Listing 14: Timer code for Binaural Beats experiment

Firstly, the output string is built with the current EEG data and a timestamp. It is written to the output *csv* file and the string is resetted. Then, the current time is subtracted by the *current_time* variable that has been set at the beginning of the stage. Finally, a short sleep time is implemented.

The core components and their key functionalities have been explained up to this point. In the last section, the device-specific bash scripts and their usage is demonstrated. Also the output format will be shown.

3. Software Usage

This section explains how the documented software can be used for the EEG devices.

3.1 Eye State Classification

On the MindWave EEG headset, the *captureEEGEye.py* component can be run as is. The listing below shows an example use case.

```
./captureEEGEye.py –folder /home/user/captures
```

Listing 15: Example Usage for capturing the eye state with the MindWave headset

This will run the experiment until the user hits the “Escape” button and will save camera captures and a single *csv* file to the folder */home/user/captures.*

On the Emotiv EPOC EEG device, the code for the headset initialization has to be removed or commented out (with the “#” character) since it is MindWave specific. Also, the string creation has to be slightly modified.

```
# rawEEG = eegCapture.RawDataPlotting(foldername)
# string = str(int(round(time.time() * 1000))) + ";" + rawEEG.getData() + camera.capture() + "\n"
string = str(int(round(time.time() * 1000))) + ";" + camera.capture() + "\n"
# rawEEG.visualize()
# rawEEG.close()
```

Listing 16: Changes to be made for the Emotiv EPOC headset

This is persisted in the *captureEEGEye_Emotiv.py* file for future use. A bash script is then used to run the EEG capturing. This is implemented in the *runExperiment.sh* file.

```
#!/bin/bash
../emotiv/ozancaglayan/python-emotiv/examples/record-data.py 135 &
python captureEEGEye_Emotiv.py --folder captures

...

./runExperiment.sh
```

Listing 17: Bash script to run the eye state classification on Emotiv EPOC

3.2 Binaural Beats Playback and Classification

The approach to the binaural beats experiment is similar to the procedure explained above. However, the MindWave experiment has to be run from a bash script as well, since the *shapecount.py* component has to be run in parallel. Therefore, two respective scripts exist, *runExperiment.sh* and *runExperiment_epoc.sh*

Comparable to the eye state classification we also have to Python core files for each device, one with the headset initialization and usage and one without. Since this is identical to the procedure above it is not described further here. In the following listings, the experiment scripts and their usage are shown.

```
#!/bin/bash
python binauralExperiment_mindwave.py out &
sleep 6
python shapecount.py
```

Listing 18: Binaural Beats on MindWave

```
#!/bin/bash
python binauralExperiment.py -f out &
sleep 6
../emotiv/ozancaglayan/python-emotiv/examples/record-data.py 770 &
python shapecount.py
```

Listing 19: Binaural Beats on Emotiv EPOC

```
./runExperiment.sh
./runExperiment_epoc.sh
```

Listing 20: Running the experiments

Note the sleep time of 6 which is necessary to have both scripts run at approximately the same time since the *binarualExperiment* script needs some time to initialize fully. It does not have to be the exact amount of time since the results are always assigned a timestamp.