computergraphics tübingen

Funfair: EEG-based game control

Robert Eisele & Robert Geirhos

April 17, 2017

Praktikum Computergrafik WS2016/17
Wilhelm Schickard Institute of Computer Science

Funfair: EEG-based game control

Robert Eisele & Robert Geirhos

Figure 1: You can place a teaser here.

Abstract

Electroencephalography (EEG) is a technique for measuring brain activity. Although widely used in neuroscientific research, it has thus far only rarely been used for the purpose of computer game control. Here we show that a leightweight consumer EEG device, the Emotiv Insight[1], can be successfully used to play a broad range of simple yet challenging computer games. Furthermore, we demonstrate that rather than merely substituting a mouse or keyboard, an EEG device enables the development of a new type of games which go beyond traditional means of game control by enhancing human computer interaction. We specifically developed six browser games, all of which can be played by controlling one's mental activity. This could be be a starting point for a new, keyboard-free gaming experience.

1 Introduction

Traditionally, computer games were played with keyboards, mice and more specialized input devices like joysticks, gamepads and steering wheels. In order to improve gaming experience, much more sophisticated human computer interaction devices were developed, like virtual reality glasses and sensory gloves. All these efforts subserve to get games as realistic as possible. Only recently, cheaper consumer EEG devices came to market, which makes it possible to use them as input devices as well. By reading the players brain activity, a much better adaption to the gamers stress level can be applied, as well as a totally new way of computer game control can be brought up. One of such devices is the Emotiv Insight EEG, which consists of enough sensors to distinguish between several emotional levels and apply multiclass classification for higher order control mechanisms.

2 Description of the Solution

We wanted to show that it's possible to use a customer EEG as the input device for a browser game and not being just a replacement of an ordinary keyboard. As such, we developed multiple small games, which can be subsumed under the topic of a funfair. We decided against one big monolithic game, as the focus would be shifted towards real game-development instead of focusing on bringing up simple models, that don't need a training phase, work for every person out of the box and have a high reaction-time in multiple game-scenarios. In

[1] `https://www.emotiv.com/insight/`

order to get the best performance out of the sensor data, we first tried several ways of integrating the Emotiv sensor into an application. We started with the official Emotiv SDK, as it is available for several computer programming languages, like C#, Java and Python. Unfortunately, the SDK isn't really stable across different operating systems, like Linux and Mac OSX. The Objective-C version was doing fine on OSX, but Linux had several other problems. But as the features that were provided under the limiting licensing model as well as the limited data rate were not satisfying, we moved on and tried to reverse engineer the formerly Firefox extension, which was provided by Emotiv until December, 2016. As the official SDK as well as the Firefox extension did not provide the data we needed, due to licensing problems and shut down of API's, we finally tried the open source reverse engineered library Emokit by OpenYou, which was initially developed to get data out of an Emotiv EPOC EEG headset, but flawlessly works with an Emotiv Insight headset as well.

The final version of the software consists of a Python application, which forks a Tornado web-application server. This setup is ideal to stream the sensory data from Emokit directly to the browser via websockets.

On the browser-side, the data is received asynchronously, both from the Emotiv data stream, as well as from the Leap Motion sensor, which was used for one of the games to bring in another way of non-standard interaction with games. The games themselves are organized in separate HTML files, which share one common JavaScript file. This JavaScript file handles the game-progress, implements a central animation loop, that can be used by every individual game as well as a status bar, which indicates the current concentration level of the player.

2.1 Sensor data analysis

All data, if not stated otherwise, were analyzed using R [**?**]. Our goal was to build several sensor data classifier which, based on an analysis of the data, would be able to tell apart several menal states (`no_action, action_1, action_2, ...`). In the process of building these classifiers, we faced two key challenges: Firstly, there was no prior mapping from the 16 sensors to a mental state - moreover, we did not even know which sensor would correspond to which mental signal (like α or β waves). Secondly, the sensor data - as is the case for almost any recorded data - seemed to be noisy. We therefore approached both challenges in a data-driven way: We recorded data for a variety of different actions and mental commands and carefully explored and visualized the sensor data recordings. As can be seen in Figure 3, some commands such as `clench fist` were barely visible in the data, whereas other commands (e.g. `shake head, clench teeth`) produced considerably consistent spike-like patterns. We therefore decided to focus on the actions that seemed to be visually distinguishable in the data.

As most actions lasted for a very brief period of time, roughly 1-2 seconds, we buffered all sensor data individually with a sliding window of 2 seconds, which corresponds to 256 data points per sensor since the sampling frequency was 128 Hz, and took this sliding window as the basis for all further data analysis. We then implemented two different types of classification. The first one is a mapping based on a combination of several sensor values to a continuous value indicating the degree of relaxation (from tense, agitated to very relaxed and calm). It was achieved by taking into account the variance of the sum of all sensor data, which is based on the observation that a very tense state cannot be traced back to one particular sensor, but elicits high variance in several sensor recordings. The second type of classification, detecting discrete events, directly focuses on specific sensors: `Y` and `X` for `shake head` classification, `F3` for `nodding` and `O1` for the classification of `clench teeth`. It would have been possible to classify `squint eyes` as well, however the signals appeared to be hardly distinguishable from `clench teeth`, and needing to close one's eyes during gameplay was found to be rather annoying. This second type of classification was achieved through thresholding the windowed sample variance of a targeted sensor. As a result, we were able to classify one continuous mental state (`relaxation`) as well as four discrete events (`clench teeth, nodding, shake head, no_action`).

2.2 Gyro data analysis

It turned out that Emokit was the perfect fit for our needs, as it gives raw measurements at a pretty high rate. Unfortunately, there is no documentation of what channel actually delivers what infor-

mation. When the headset is moved around, a lot of channels dramatically change their values, which must be kept in mind for normal sensor analysis.

2.3 Architecture & Backend

2.4 Highstriker

A highstriker is a classic game that can be found on most funfairs. We took the basic idea of striking with a hammer to achieve a score on a vertical bar and made a few adaptions to it. Rather than a using a virtual hammer, we mapped the current mental state to a score: the more relaxed one is, the better the score (and the friendlier the status indicator, ranging from `Burnout candidate` to `Master of Yoga`). As this game is, in our experience, both quite fun and very easy to play, we selected it to be the first game, enabling the user to get to know how he can control his mental activity to achieve a good score. This then prepares for more complex games such as the balancing game. So-called *brain self-regulation* based on visual feedback has been successfully used in clinical settings, e.g. to teach psychopaths to control their aggression [**?**]. It would be interesting to see whether some of our games (the ones that can only be won when one learns how to calm down and relax quickly) could be used to train children suffering from ADHD (attention deficit hyperactivity disorder) in a fun way.

2.5 Painter

The core idea behind the painter game can be described as follows: Wouldn't it be fun to take a snapshot of yourself (using your computer's camera) and then having this image painted in a style that matches your mood?

We achieved this by accessing the computer's camera through the browser, sending the plain image to the backend which forwards it, along with a style inferred from the current mood, to the Turbo-Deepart website[2] via the DeepArt API[3] to paint the image, which is then subsequently sent back to the game. DeepArt basically implements a neural network-based technique for painting an input image in the style of an arbitrary style image [**?**]. For an illustration of how the Lena image looks like when painted in four different styles, see Fig. 2. We were able to contribute a commit to the DeepArt API (`rgeirhos`, commit `50f200f`, merged Dec 14, 2016), solving an early image retrieval issue.

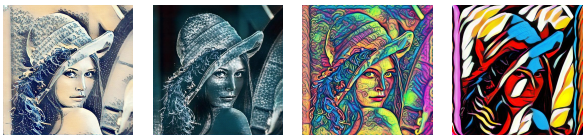

Figure 2: Four different style examples for the painter game. Images are sorted from a relaxed, calm style (left) to a more excited, agitated style (right).

As an additional feature, the painted image is not only shown but lies behind a veil first (low alpha value). One then needs to use either the mouse to hover over the image and reveal it piece by piece, or use a Leap Motion controller[4] to do so. A Leap Motion device measures hand gestures and translates them into a 3D hand model on the screen. We mapped the position of the hand model to a certain position on the canvas, which enables the user to simulate holding a paint brush in his or her hands and paint the image with painting-like movements.

2.6 Magic Duel

2.7 Mastermind

Mastermind is a board game where one player selects a four-digit, order-sensitive color code and the other player tries to infer this code based on the feedback he receives in several rounds of guessing: Basically, for every token in the correct position (i.e. matching the code's color at this position), one receives a black feedback token; for every correctly colored token (that is, the color appears in the code) in the wrong position, one receives a white feedback token. This game serves as a demonstration that also games with four different control events can be controlled with the Emotiv device. The full list of game commands (in brackets: corresponding mental commands) is: change row (`nod`), change column (`shake head`), change color (`clench teeth`) and finally do nothing (every other mental state / action). In our experience, it is quite interesting to play this game with head movements and mental commands for the first time, however after a while, one would like to use a less cumbersome mouse instead. As this was not the case for other games, we think this might point to the fact

[2] `http://turbo.deepart.io/`
[3] `https://github.com/deepart-io/deepart-api`
[4] `https://www.leapmotion.com/`

that using an EEG device to replace a mouse is a bit boring - after all, we are pretty good in controlling computers with mouse and keyboard on a daily basis, whereas games that can only be played with such a device seem to be way more interesting.

2.8 Wireloop

Wireloop is a remake of the classical game whereby a metal loop has to be dragged from one end to the other, without touching the wire. Typically a loud sound is played when the wire is touched accidentally. In the Funfair setting, the wireloop game is controlled by the gyroscope only.

By moving the head, the rotational change is translated into planar co-ordinates that enable the metal loop to move. The wire itself is generated randomly using cardinal splines and is fully parametrable. The game is restarted whenever the wire was touched. One simplification was made. The angle of the loop is automatically adapted to the current position by taking the derivative of the current line segment and is not needed to be set by the player.

2.9 Balancing

Our balancing game is perhaps the most challenging game, as it requires to balance a little man on an inherently instable seesaw: if tilted to one side, one needs to strain oneself in order to get it back to balance - but then, it tilts to the other side and one instantly needs to switch to 'zen mode', to total relaxation... if one is able to do this for 30 seconds, the game is won, otherwise it is re-started.

In some relaxation techniques such as *Progressive Relaxation* [**?**], being able to consciously change from straining oneself to relaxation mode (and vice versa) is a core element: if one learns and trains how to change between these states, this can be effortlessly applied in situations where one is naturally stressed and tense, such as ahead of an exam or a job interview. It would be interesting to explore whether a game like our balancing game could be used in a similar fashion - which would then enable one to teach relaxation techniques to a broader variety of people than those attracted by, say, evening classes.

3 Possible Extensions

Describe possible extensions and discuss why they could be useful.

Viele Daten -> LSTM
weitere Spiele beliebig hinzufuegbar
Spiele fancier machen (3D, ...)

A General Infos

You don't need an appendix. The two appendix sections are just there to give some additional or general information.

This document describes roughly, what the documentation of the Praktikum project should look like.

Note that your final documentation of your project should contain 6 text pages using this template plus the cover and the empty sheet at the beginning. Within the 6 pages, at least 4 pages should be only text.

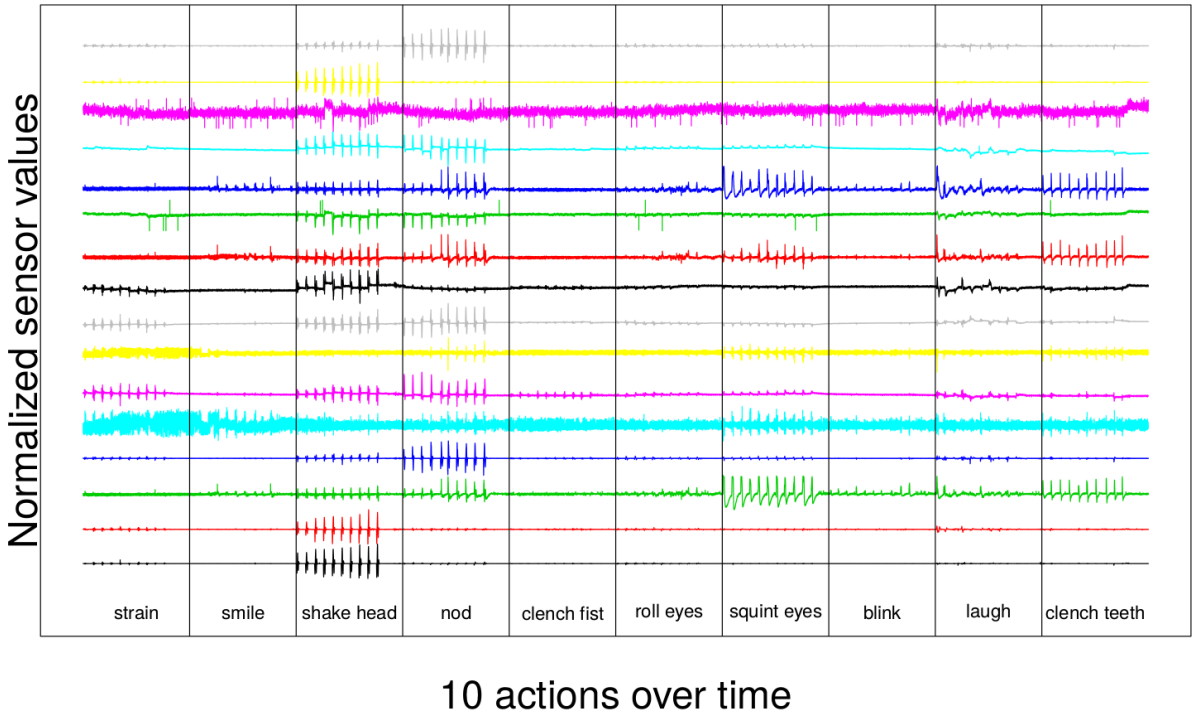

Figure 3: Raw sensor data for ten different actions and mental commands. All data are normalized to lie within [0, 1]. Every sensor has a different offset here for better visibility. Sensors from top to bottom: `Y, X, F4, FC6, AF4, F8, T8, P8, O2, O1, P7, T7, F7, AF3, FC5, F3`. Every action corresponds to one minute of data recording, during which over the first 45 seconds the action was briefly executed every five seconds, followed by a short break until the next type of action began.