FAKULTÄT FÜR !NFORMATIK

Building an auditory and visual stimulation device with EEG feedback loop

An open source software and hardware approach for controllable brainwave entrainment

Bachelor thesis

for obtaining the academic degree

Bachelor of Science

as part of the study

Computer Engineering

carried out by

Robert Annessi

student number 0527718

Department of Computer Engineering

at Vienna University of Technology

Supervision:

Dipl.-Ing. Dr.techn. Christian El-Salloum

Vienna in May 2010

Author's signature

Supervisor's signature

Abstract

Altering their state of consciousness is a frequent wish of many people. To accomplish this, some train through meditation or similar techniques over many years or even decades. Others even have problems to reach common states like falling asleep. Is the state of consciousness controllable in an automatic way achieving similar results as training over years of meditation?
In the following approach the subjects are stimulated with light and sound which results in evoked potentials that are measured by an electroencephalograph (EEG). An auditory and visual stimulation device and the ModularEEG of the OpenEEG project are built whereby only open source software and hardware is used. The data gathered from the EEG is split into frequency components and analyzed in real-time while the subject is continuously stimulated with light pulses and binaural beats at certain frequencies. Due to the frequency following response phenomenon it has been possible during tests to let subjects fall asleep and wake up again purposely.
So it may even be possible to reach other states of consciousness such as concentration, relaxation or meditation as well. The control value has been set manually during the tests but results show that it should be feasible to implement an closed-loop control system for brainwave entrainment. For statistically significant statements further systematic testing has to be done, since this approach has been tested with too few subjects.

Contents

List of Figures

List of Tables

1 Introduction

Altering their state of consciousness is a frequent wish of many people. State of consciousness in this case means for example relaxation, meditation, alertness, concentration or sleep. To reach certain states like meditation, some train over many years or even decades. Others may even have problems to reach common states like falling asleep. In the following approach an auditory and visual stimulation device is built wherewith subjects are stimulated with light and sound and its effect measured by an EEG.

Short historical overview over EEG and auditory and visual stimulation devices (AVSD).

Since Hans Berger found that the electrical activity of the human brain is measurable from the scalp electroencephalography evolved a lot. It is used for medical purposes such as monitoring coma or brain death, investigate sleep disorders or test epilepsy among others.
Stimulation of the human brain by auditory and visual stimuli has a long history in humanity. "*Ancient scientists were fascinated by the phenomenon of flickering lights. Apuleius experimented in 125 A.D. with the flickering light produced by the rotation of a potter's wheel, finding that it could reveal a type of epilepsy. Ptolemy studied in 200 A.D. the phenomenon of the flickering generated by sunlight through the spokes of a spinning wheel. He noted that patterns and colors appeared in the eyes of the observer and that a feeling of euphoria could be experienced.*" [4]
"*Moreover, combinations of rhythmic sound and flickering light were used by the ancients to induce trance.*" [4]
"*In the 1930s and 40s scientists such as W. Gray Walter and others used powerful electronic strobe lights and the new EEG equipment to alter brainwave activity producing trance-like states of profound relaxation and vivid imagery.*" [4]

Nowadays AVSD are used for medical purposes such as the treatment of attention deficit disorder, depression, epilepsy or alcoholism. [12, p. 5]
"*Some researchers assume that subjects can improve their mental performance, normalize behaviour, and stabilize mood through the positive or negative feedback loop, while others are sceptical on these controversial issues. There are some findings indicating applications to certain range of conditions, as attention deficit disorder, depression, epilepsy, and alcoholism.*" [12, p. 5]

Problem statement

Is it possible to measure the effect of sound and light pulses on the human brain with an open source software and hardware EEG? If effects can be measured, is it possible to build a closed-loop control system by setting a target frequency and make the subject eventually reach that frequency through light pulses and audio tones whose frequency is based on the EEG measurements? Is it possible to build an open source software and hardware AVSD and EEG to implement that loop? Is it possible to do the analysis for that loop in real-time or is the computational power needed currently to high?

Objective

The subjects are stimulated with light and sound which results in evoked potentials that are measured by an electroencephalograph (EEG). An auditory and visual stimulation device and the ModularEEG of the OpenEEG project are built whereby only open source software and hardware is used. The data gathered from the EEG is split into frequency components and analyzed in real-time while the subject is continuously stimulated with light pulses and binaural beats at certain frequencies.

Structure of the paper

Section 2 includes the theoretical background of AVSD, electrodes and their usage, electroencephalography, brainwave entrainment and visual and auditory evoked potentials needed to understand the approach described in this paper.

Section 3 includes the evaluation of AVSD, EEG and a detailed description of the ModularEEG and electrodes. Also the setup of the ModularEEG is described such as building the boards, the board cabling, building the electrodes, electrode cabling and head mounting the electrodes. Furthermore EEG software is evaluated.

Section 4 describes the signal evaluation, the test setup and the results gained.

Section 5 includes the discussion.

2 Theoretical background

This section includes the theoretical background used for the approach described in this paper. The basic concepts of AVSD, active and passive electrodes and their usage, EEG, brainwave entrainment, visual and auditory evoked potentials.

2.1 AVSD

Auditory and visual stimulation devices are often called "Mind machines", "Dream machines" or "Sound and Light-Machines".
They are usually quite small devices that consist of a pair of headphones, light goggles and a unit which controls the Light Emitting Diodes (LED) and the sound. Light and sound pulses at a certain frequency to alter the brains electrical oscillations which is described in the evoked potential section. The viewer experiences complex colored patterns while his eyes are closed.

2.2 Electroencephalography

"*Electroencephalography (EEG) is the recording of electrical activity along the scalp produced by the firing of neurons within the brain. In clinical contexts, EEG refers to the recording of the brain's spontaneous electrical activity over a short period of time.*" [14, 1258 pp.]
In 1924 Hans Berger, a German neurologist, developed a method to gather the electrical activity of the brain from the scalp with ordinary radio equipment. [3]
Electrical oscillations are derived bipolar or monopolar. The signal is then amplified, digitalized and finally recorded and/or processed further. The time resolution of the EEG recording is determined by the sampling rate which indicates the measurements by seconds. The sampling amplitude is usually within 1 to 100 µV. The resolution in space is determined by the number of electrodes which depends on the device. [22, p.103-105][15, p. 44]
When representing EEG recordings in the frequency domain you get the frequency and power spectrum. The common convention for the frequency bands is shown in table 1. [22, p. 102] There are slight differences in the frequency ranges across publications.

Name	Frequency band	Occurrence
Delta (δ)	0.5 - 4 Hz	During sleep
Theta (θ)	4 - 8 Hz	Near-unconscious states (e.g. just before waking or sleeping)
Alpha (α)	8 - 13 Hz	Wakeful relaxation with closed eyes
Beta (β)	13 - 30 Hz	Normal waking consciousness
Gamma (γ)	25 - 100+ Hz	Not clear

Table 1: EEG frequency band comparison table

All electrical oscillations whose source is not the brain but are recorded anyway are called artifacts which have to be considered when analyzing data from the EEG. Biological artifacts are produced by the subject when moving its eyes, breathing, heart beats or other muscle activity. Most of them can not be avoided but minimized. Besides technical failures such as 50 Hz or 60 Hz noise of the alternating current can be part of the EEG recording which generally can be avoided. [22, p. 653-685][15, p. 46-48]
EEG can be seen as a non-steady, highly complex process that represents the cognitive state of the brain. This process is characterized by distributions through space and frequency spectrum. Thought processes can be distinguished by the activity in certain areas of the cortex in a certain frequency range. The EEG signal first has to be transformed from time into frequency domain with the Fourier transformation to analyze the frequency spectrum. With computer aided spectral analysis the Fast Fourier Transformation (FFT) is being used since computers can calculate that notably fast. [15, p. 50-52]
It should be noted that only about 1/3 of the brain's electrical activity is measurable from the scalp. [22, p. 54]

As already mentioned two types of measurements can be distinguished:

- Monopolar
- Bipolar

When doing bipolar measurement two electrodes are used for each measurement channel and each electrode has its own reference electrode. When doing monopolar measurement one shared reference electrode is used. The main advantage of monopolar measurement is that you gather a measurement channel for each electrode but the shared reference ($n - 1$ measurement channels overall). Its main disadvantage is that the shared reference has great impact on the measurement as a whole. If problems such as bad contact occur to the shared reference, all measurement values are affected negatively.

When measuring bipolar one bad electrode only affects a single measurement channel but there are only $\frac{n-1}{2}$ measurement channels available overall.

2.3 Electrodes

Electrodes are used to pick up electrical activity from the scalp surface.
Two types of electrodes can be distinguished by their way of amplification:

- **Active** electrodes
  which have some built-in circuitry that amplifies the electrical current and and therefor should improve the signal quality
- **Passive** electrodes
  which have no built-in circuitry.

Placement

The placement of electrodes is very important for being able to reproduce and compare test results over time and subjects. Therefor the "international 10-20 system" was developed by Herbert Henri Jasper in 1958. It is used to describe and apply the location of electrodes on the scalp. [14, p. 140] A graphical representation is shown in figure 1. [13, p. 54]
"*This system is based on the relationship between the location of an electrode and the underlying area of cerebral cortex. The "10" and "20" refer to the fact that the actual distances between adjacent electrodes are either 10% or 20% of the total front-back or right-left distance of the skull.*" [17]
The quality of the acquired signal highly depends on good contact between the scalp and the electrodes. To ensure good contact conductive paste should be used.
As previously mentioned he EEG electrodes do not only collect electromagnetic signals from the brain but also a lot of noise. The most important source of noise comes from power lines with oscillations of electromagnetic signals at 50 or 60 Hz. Unfortunately the noise is far greater than the typical EEG signal.
To address this problem the EEG amplifiers measure two electrodes which both record the same noise but different EEG signal since they are placed on different positions on the scalp. The amplifier now subtracts the two signals, thus canceling out the noise. In an ideal world only the EEG signal would be left. In general, we can say that the smaller the impedance the lower the noise signal. To achieve very low impedance the application site should be cleaned by using a adhesive gel. Knowing that the noise signal depends on the impedance and that both the noise of the signals must be exactly the same, we understand that the adequacy of both impedances is even more important than the impedance of the values themselves.

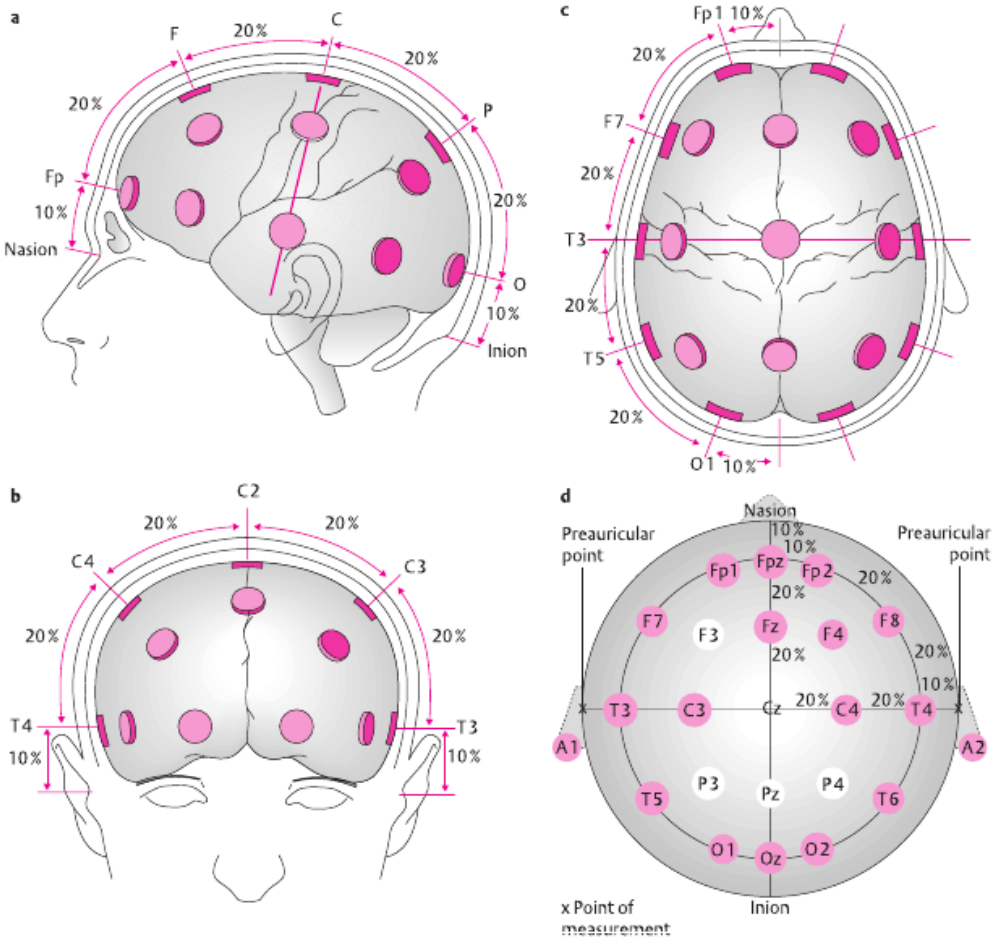


Figure 1: International 10-20 system; name and position of electrodes

[7] In order to prevent signal distortions the impedance of each electrode should be below 5 kΩ and balanced within 1 kΩ of each other. [12, p. 7]

2.4 Brainwave entrainment

"*Entrainment describes a process whereby two rhythmic processes interact with each other in such a way that they adjust towards and eventually 'lock in' to a common phase and/or periodicity.*" [5, p. 2]
"*It [R.A.: Brainwave entrainment] depends upon a "frequency following" response, a naturally occurring phenomenon where the human brain has a tendency to change its dominant EEG frequency towards the frequency of a dominant external stimulus.*" [18] It is more effective if the stimulus frequency is near to the

subjects current dominant frequency. [21]

Visual brainwave entrainment

Eyes are stimulated with light pulsing at a certain frequency. It is possible to stimulate both eyes synchronously as well as alternating. When using alternating pulsing lights it is important that one eye can not see the light pulses designated to the other eye.

Auditory brainwave entrainment

"*Binaural beats or binaural tones are auditory processing artifacts, or apparent sounds, the perception of which arises in the brain for specific physical stimuli. This effect was discovered in 1839 by Heinrich Wilhelm Dove.*
The brain produces a phenomenon resulting in low-frequency pulsations in the loudness and sound localization of a perceived sound when two tones at slightly different frequencies are presented separately, one to each of a subject's ears, using stereo headphones. A beating tone will be perceived, as if the two tones mixed naturally, out of the brain. The frequency of the tones must be below about 1,000 to 1,500 hertz for the beating to be heard. The difference between the two frequencies must be small (below about 30 Hz) for the effect to occur;" [21]
The resulting beating tone affects the state of consciousness of the subject. [10] There is little to no evidence that combining visual and auditory stimulation does increase the effect. [9, p. 13]

2.5 Evoked Potential

An evoked potential is the electrical response of the brain to a sensory stimulus. It tends to be low, ranging from less than a micro volt to several micro volts, compared to tens of micro volts for EEG. To resolve these low-amplitude potentials against the background of ongoing electrical signals, signal averaging is used. The signal is time-locked to the stimulus and most of the noise occurs randomly, allowing the noise to be averaged out with averaging of repeated responses. [12]

Visual Evoked Potential

"*Visual evoked potential (VEP) is caused by sensory stimulation of a subject's visual field and is observed using electroencephalography. Commonly used visual stimuli are flashing lights or checkerboards on a video screen that flicker between black on white to white on black (invert contrast).*" [19]

Auditory Evoked Potential

Auditory evoked potentials are very small electrical voltage potentials originating from the brain recorded from the scalp in response to an auditory stimulus, such as different tones, speech sounds, etc. [19]

3 Approach

This section includes the evaluation of AVSD, EEG and a detailed description of the ModularEEG and electrodes. Also the setup of the ModularEEG is described such as building the boards, the board cabling, building the electrodes, electrode cabling and head mounting the electrodes. Furthermore EEG software is evaluated.

3.1 AVSD

First approach

My first approach to AVSD was the brain machine[1] designed by Mitch Altman who published it in MAKE Magazine Volume 10: Home Electronics. You can find the schematics and firmware for it on its website.
Although it is a very interesting project to start with it does not fit the following requirements for this approach:

- Universal Synchronous/Asynchronous Receiver/Transmitter (USART) communication which is needed for sending commands to the device
- Reasonable sound quality
- Volume control

I could not find any commercial AVSD that is open source software and hardware. Besides such a device is fairly easy to build.

Latest approach

In the latest approach the sound is generated by a PC and the light pulses by a microcontroller.
The Python script "mindcommander.py" which is listed in the appendix runs on the PC and reads data from standard input (stdin) where values from 02 to 29 are accepted. [2] If it receives a valid value it plays a tone at 200 Hz from a Waveform Audio File Format (WAV) file on the left audio channel and another tone at 200 Hz plus that offset read from stdin on the right audio channel to make a pulsing

[1] http://makezine.com/10/brainwave/

[2] These values represent the frequency the subject is going to be stimulated at. In the approach described in this paper only the frequency band ranging from high Theta to Beta is important. Therefor the frequency range from 2 to 29 is totally sufficient. Even though it could be extended both in range as well as in granularity.

sound perceived by the subject. First I used sounds around 500 Hz, but 200 Hz sounded more comfortable for some subjects during tests. Anyway it would be possible to use other values as mentioned previously. Besides the script sends that value to a microcontroller over USART and listens if it sends the same value back to acknowledge the receiving of the data.
For generating the light pulses the Arduino board - "*an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software*" - is used. [1] Figure 2 shows the visual stimulation device built for the latest approach. It reads two bytes over USART which are two digits from 02 to 29 that represent

Figure 2: Visual stimulation device

a frequency value in Hz. If these two bytes are received it sends that data back and updates the timer top value to trigger an interrupt at the given frequency. The corresponding interrupt service routine (ISR) then toggles the state of both LEDs. In the first approaches the LEDs were blinking synchronously whereby in the latest approach the LEDs blink alternately. The source code for the microcontroller "mindcontroller.c" is listed in the appendix.

3.2 EEG

There are different approaches to build an open source software and hardware EEG. The OpenEEG[3] project currently consists of two sub-projects: Modula-

[3] http://openeeg.sourceforge.net/

rEEG[4] and soundcardEEG[5] which have different approaches. Besides OpenEEG there are other projects such as the MonolithEEG[6] or the OpenEXG-2[7]. They all have in common that they only provide two channels. Only the ModularEEG can be extended to support a maximum of six channels. In this approach the ModularEEG is used.

3.2.1 ModularEEG

The ModularEEG consists of one analog board to which the electrodes are connected and a digital board that handles the signal capturing and data sending over USART.

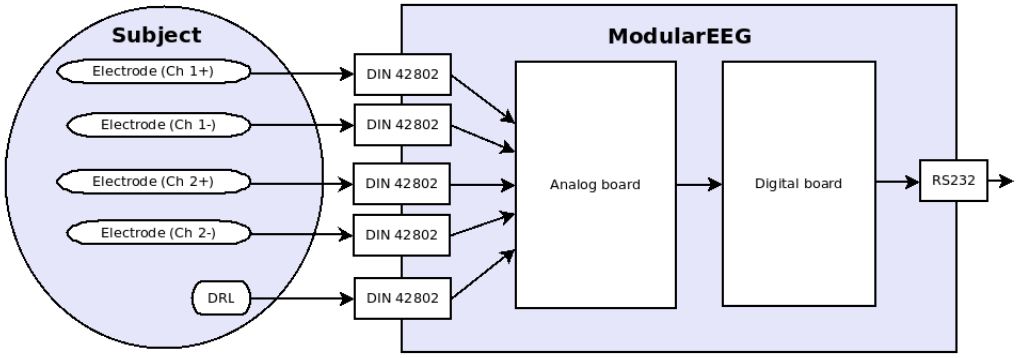


Figure 3: ModularEEG overview

Figure 3 shows the overview of the ModularEEG. There are two electrodes for each channel since the ModularEEG measures bipolar. The other input is for the Driven Right Leg (DRL) which is connected to an ear lobe and used for noise cancellation: primarily the mains hum as previously described in section 2.3. The digital board sends the digitized EEG data through a standard serial cable for further processing.
There are bare and preassembled boards available from Olimex Ltd[8] for both two and six channel versions. The preassembled boards are already flashed with the latest firmware and ready for use. Buying preassembled boards saves a lot of work and is only a little more expensive then building your own.

[4] http://openeeg.sourceforge.net/doc/modeeg/modeeg.html
[5] http://openeeg.sourceforge.net/doc/hw/sceeg/
[6] http://freenet-homepage.de/moosec/projekte/simpleeeg/index-Dateien/Page431.htm
[7] https://sites.google.com/site/openexg/
[8] http://www.olimex.com/

The schematics for the analog and digital board are available at the ModularEEG website[9].

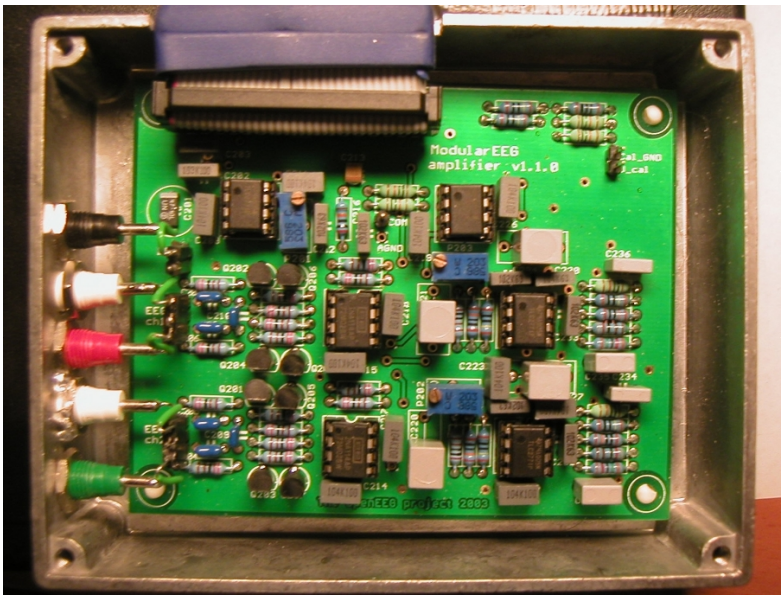


Figure 4: ModularEEG analog board

Figures 4 and 5 show the analog and digital of the ModularEEG.

Cabling

Two types of cables are needed for the ModularEEG:

- A cable for connecting the analog and the digital board:
  "*The board-to-board cable is a 34-lead ribbon cable. You can make one from an old floppy-disk cable. It is recommended that you buy a new female connector so that you can make it shorter if you want to. The distributors that sell electronic parts usually have connectors as well.*" [6]

  It is important to use a straight cable. Floppy cables usually have one straight and one twisted side.

- Cable for the serial connection to the PC:
  Details on how to build the cable for the serial connection to the PC are listed in the schematics file in the appendix.

[9] http://openeeg.sourceforge.net/doc/modeeg/modeeg.html

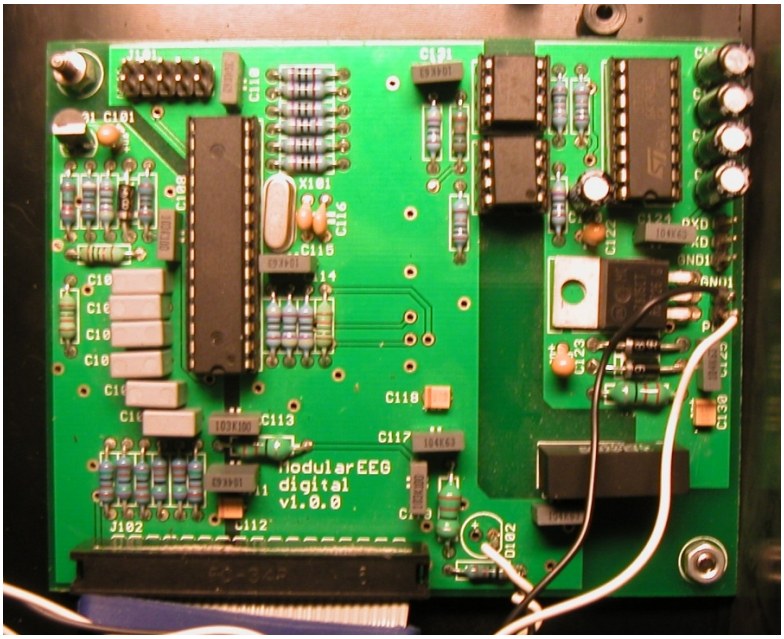


Figure 5: ModularEEG digital board

Casing

Two cases are needed for the ModularEEG:

- A metal case for the analog board which acts as electromagnetic shield.
- A plastic case for holding the metal case, the digital board and the connector cable between them.

The cable connecting the analog and digital board also needs to fit in the plastic case. A power switch, a power state LED, plugs for the electrodes and a power connector are also built-in. A very well fitting metal case from rs-components[10] is used in this approach (order number 343-9899). Unfortunately the plastic case used does not fit well.

3.3 Electrodes

3.3.1 Construction guides

There are a few construction guides for active electrodes available on the internet:

[10] http://www.rs-components.com/

- Jarek Foltynski[11]
- Joe Street[12]
- Jim Peters[13]
- Pedro Ortega[14]
- Radek Panek[15]
- Adam Libura[16]

Table 2 compares the electrode guides. Unfortunately most of the above guides are a few years old and do not offer complete information. The amplifier circuits described in the guides are quite similar but the shape of the electrodes and their cabling and casing are implemented totally different. One guide uses Surface Mount Device (SMD) design while the others use the through-hole design which is easier to assemble.

Author	Electrode type	Power supply	Circuit design
Joe Street	Needle	Batteries	through-hole
Jim Peters	Needle	ModularEEG	through-hole
Pedro Ortega	Needle	ModularEEG	through-hole
Radek Panek	Rod	ModularEEG	SMD
Adam Libura	Wire	ModularEEG	through-hole

Table 2: Comparison of electrode guides

3.3.2 Pick-ups

Material

Electrodes for EEG measurement can be made out of different materials. Typically silver or gold is used whereas silver is said to perform better. Silver/Silver-chloride (Ag/AgCl) electrodes produce even better results.
It is possible to build electrodes yourself. Typically you use silver as base material. To further improve the signal quality you can chloride the silver electrodes which

[11] http://www.bioera.net/bw/ae/
[12] http://openeeg.sourceforge.net/doc/hw/joe_ae/
[13] http://uazu.net/eeg/ae.html
[14] http://www.dcc.uchile.cl/~peortega/ae/
[15] http://radek.superhost.pl/active_electrodes/
[16] http://www.bioera.net/bw/ae/adam/index.html

can be done with electrolysis. Sintered Ag/AgCl electrodes do even deliver better signal quality but the downside of Ag/AgCl is that it is hardly possible to work on form that sintered material. To keep the Ag/AgCl electrodes in a working shape, care has to be taken. See the usage instructions listed in the appendix.

Connector

Even though the ModularEEG website suggests to use mini-XLR plugs, the de facto standard in EEG measurement is DIN 42802.

3.3.3 Head mounting

When using a small number of measurement channels as with the ModularEEG, the positioning of electrodes on the head is very important, since the location of the signal is highly depending on the part of the brain where it is generated.
In this approach the occipital and the parietal lobe are of interest. The occipital lobe is the visual processing center of the human brain. The parietal lobe is known to receive input from a number of senses including auditory and visual. So according to the international 10-20 system, 01-02 is used for one measurement channel and 03-04 for the other.
A simple impedance tester published by Ian McCulloch that uses audio input is listed in the appendix.

3.3.4 Cabling

For each electrode one lead is needed for the signal and another two leads for the power supply, if active electrodes are used. If a four-lead cable is used, the cabling can be done within a single cable for each bipolar channel. For the connector a TRS connector (tip, ring, sleeve) can be used.

3.3.5 Conclusion

After fiddling around a long time with building and using active electrodes I definitely would recommend buying commercial passive electrodes with DIN 42802 plugs with 1.5 mm diameter. Figure 6 shows two electrodes with one being connected to the ModularEEG.

3.4 EEG Software

Hard requirements on the EEG software for this approach:

Figure 6: Electrodes connected to the ModularEEG

- Must be able to gather data from the ModularEEG (modEEGp2 and/or modEEGp3 protocol)
- Must be able to record data from the ModularEEG
- Must be able to play prerecorded data from the ModularEEG
- Must be freely available. Both free of charge and access to source code

Soft requirements on the EEG software for this approach:

- Should be able to do the analysis of the EEG data
- Should run under Linux

The OpenEEG website[17] lists some software to analyze EEG data:

[17] http://openeeg.sourceforge.net/doc/sw/

- NeuroServer[18]
  which is from 2004 licensed under the GNU Lesser General Public License (LGPL) but seems to be unmaintained. I was able to connect to the ModularEEG but the connection got interrupted within a few minutes. Hence it was not stable enough for me to use.

- BioEra[19]
  is a commercial Microsoft Windows only software which therefor does not meet the hard requirements.

- BrainBay[20]
  is an open source Microsoft Windows only software licensed under the GNU General Public License (GPL) which also runs under Linux almost full featured with the help of Wine[21] which allows "*Unix-like computer operating systems to execute programs written for Microsoft Windows*". [20] It is working really well with the ModularEEG. Most of the EEG data analysis for this approach is done with this software.

- BWView[22]
  is an open source software licensed under the GPL for viewing EEG recordings. It is available for Linux and Microsoft Windows. Unfortunately it is for viewing recorded data only. So it is not easily possible to both record and view EEG data in real time.

- EEGMIR[23]
  is licensed under the GPL and can be used for both viewing recorded and live EEG data. It is available for Linux and Microsoft Windows. Unfortunately it is currently unmaintained.

- ElectricGuru[24]
  is a freely available but closed source Microsoft Windows only software and therefor does not meet the requirements.

- BioExplorer[25]
  is a closed source Microsoft Windows only software and therefor does not meet the requirements.

[18] http://openeeg.sourceforge.net/doc/sw/NeuroServer/
[19] http://www.bioera.net/
[20] http://www.shifz.org/brainbay/
[21] http://www.winehq.org/
[22] http://uazu.net/bwview/
[23] http://uazu.net/eegmir/
[24] http://www.realization.org/page/topics/electric_guru.htm
[25] http://www.cyberevolution.com/

- OpenVIBE[26]
  is an open source software which runs under Linux and Microsoft Windows. It is available under the terms of the Lesser GNU General Public License (LGPL). Unfortunately it just got support for the ModularEEG while writing this paper. Christoph Veigl finished the connection to the ModularEEG which is available since version 0.4.0.

[26] http://openvibe.inria.fr/

4 Experimental Setup

This section includes the description of the experimental setup and the measured results.

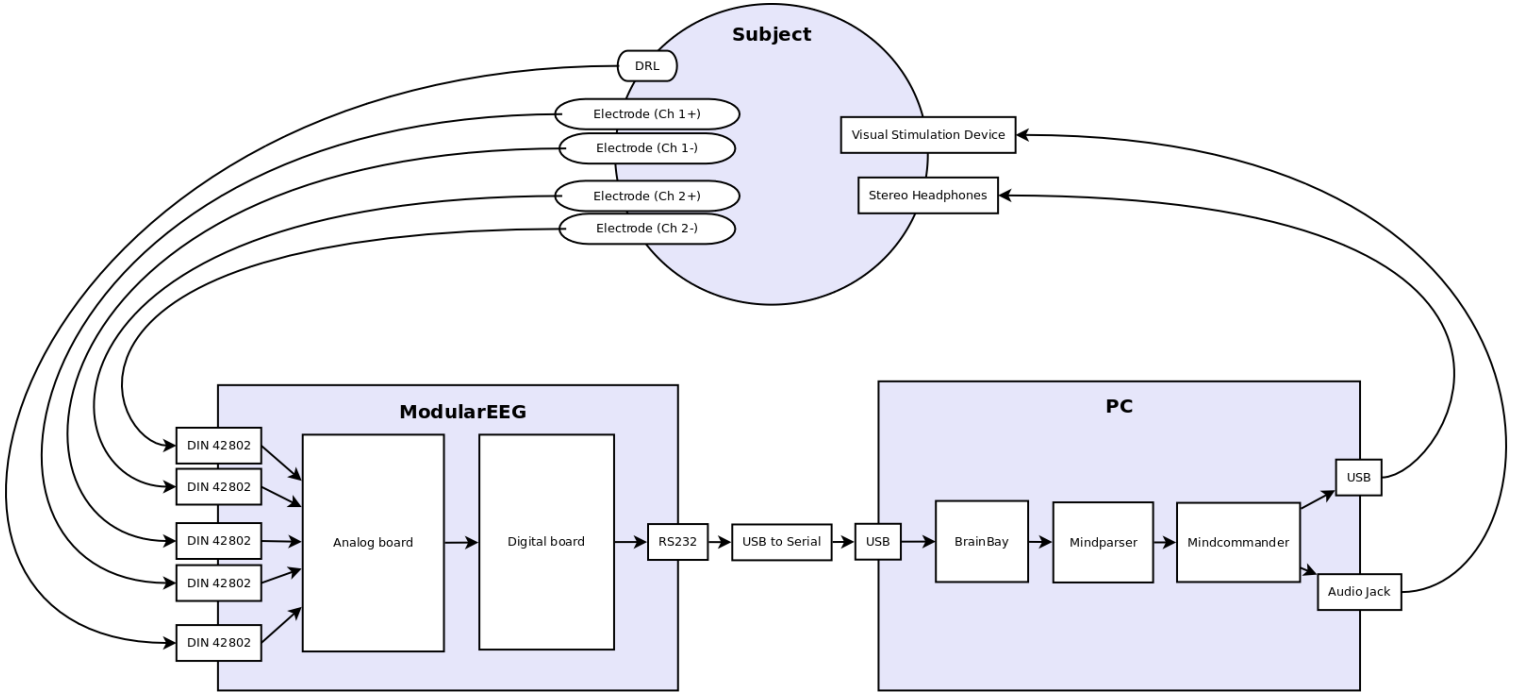


Figure 7: Big picture

4.1 Setup

4.1.1 Connections

Electrodes are connected to the ModularEEG through DIN 42802 plugs. The ModularEEG is connected to a USB port on the PC through a USB to serial adapter. The visual stimulation device (VSD) is connected to another USB port on the PC. Stereo Headphones are connected to the audio output of the PC. The subject is told to put on the goggles and headphones.

4.1.2 Subject

Preparation

The subject has been informed about the test procedure beforehand, is told not to move, to close their eyes and relax.

Electrode mounting

Passive Ag/AgCl-electrodes were mounted on the scalp with conductive paste at O1-O2 for one and C3-C4 for the other measurement channel, according to the international 10-20 system.

4.1.3 ModularEEG

The ModularEEG has been set up as described previously and programmed with firmware version v0.5.4-p2 which is listed in the appendix.

4.1.4 PC

BrainBay

A slightly modified version of BrainBay is started through Wine and the configuration file is loaded. Both the configuration file and the patches can be found in the appendix. The European Data Format (EDF)- and file-writer-modules are configured to write to files and started. BrainBay then gathers the packets received over USART from the ModularEEG. For both channels the signal is sent through a band-pass filter. Frequencies below 1 and over 30 Hz are filtered. In the next step the signal is parallelized and each filtered through another band-pass filter [27] frequency components in the range from 1-3, 2-4, 3-5, .., 28-30 Hz. On each of the separated frequency component the Fast Fourier Transformation is applied, so that the power level of each frequency band can be written to a file.

[27] A filter that only lets frequencies within a certain frequency range pass.

This is necessary because the dominant frequency calculation can not be done within BrainBay at the time of writing. Since the sampling rate of the ModularEEG is 256 Hz, 256 values for each available channel are written to the file every second.

"mindparser.py"

The script "mindparser.py" which is available in the appendix is started from within a terminal with the BrainBay output file given as argument. It parses the output file written by BrainBay and writes to *Standard Output* (stdout) at regular intervals that can be set in the script. The output of the script can then be used to set the next target frequency.

Dominant frequency calculation

Since the input data from the EEG is not precise by its nature, the data needs to be averaged. "mindparser.py" loads the stream of values representing the current power level of each frequency band into an array of a predefined size. The size of the array determines the time a value is being held in the array. The array is treated as a FIFO (First In, First Out) storage. New values are written to the array as they are gathered and old values are removed. So new values instantly affect the median value but are attenuated by older values. The median values of the power levels stored for each frequency band are printed if the predefined timeout is reached or *Ctrl+d* is pressed on the keyboard. The median has been chosen instead of the average value since it is less prone to distortions.
So changes to the median power level values can be monitored and used to set the next frequency for the light pulses and audio tones. This value could also be used to implement a closed-loop so a desired frequency value can be reached automatically.

"mindcommander.py"

The script "mindcommander.py" is also started from within a terminal with the serial device given as argument. It reads an integer value from stdin whereas the integer must be between 02 and 29. Then it sends that value through USART to the microcontroller and plays two sound files: one at 200 Hz on the left audio channel and the other at 200+value Hz on the right audio channel.

4.1.5 Generation of light pulses and audio tones

Sound

Sine wave tones at discrete frequencies ranging from 203 to 229 Hz were previously generated and saved as WAV files on the PC using Audacity[28] a "*free, open source software for recording and editing sounds*". [2]
The Python script *mindcommander.py* which is listed in the appendix reads a value from 02 to 29 from *Standard Input* and plays the corresponding sound file. Besides it sends the value over USART to the AVSD.

Light

The VSD is connected to the PC over a Type-A/Type-B USB cable and has been programmed with the firmware listed in the appendix. The LEDs in the goggles are connected to pin 1, 2 and ground on the Arduino which was described in section 3.1.

4.2 Results

There is a connection between the stimulation frequency and its median power level. An increase of its median power level can be seen with a very short delay due to averaging the values over time. The increase can be seen when the stimulation at that frequency starts and decreases again when the stimulation stops.

Procedure of the approach:

1. The subject is told to close its eyes and to relax.
2. The dominant frequency is calculated.
3. Stimulation starts at the subjects dominant frequency.
4. The stimulation frequency is decreased by 1 Hz until the subject is being in a stage of light sleep at around 5-6 Hz. When subject is asleep the stimulation frequency is increased by 1 Hz instead of decreased.
5. The previous step is repeated as soon as the power level of the stimulation frequency has heavily increased and the power levels of the other frequencies have decreased.

[28] http://audacity.sourceforge.net/

Figure 8 compares the change of the median value of the power level of each frequency band over time to the stimulation frequency. It shows that the power level of same frequency that the subject has been stimulated increases while the power level of the other frequencies decrease. During the test the subject has been in a state of light sleep. Interestingly it also shows that the power level of twice of the stimulation frequency is also increased.
The EEG data for that test is listed in the appendix. The source code for generating the plot is listed in the appendix.

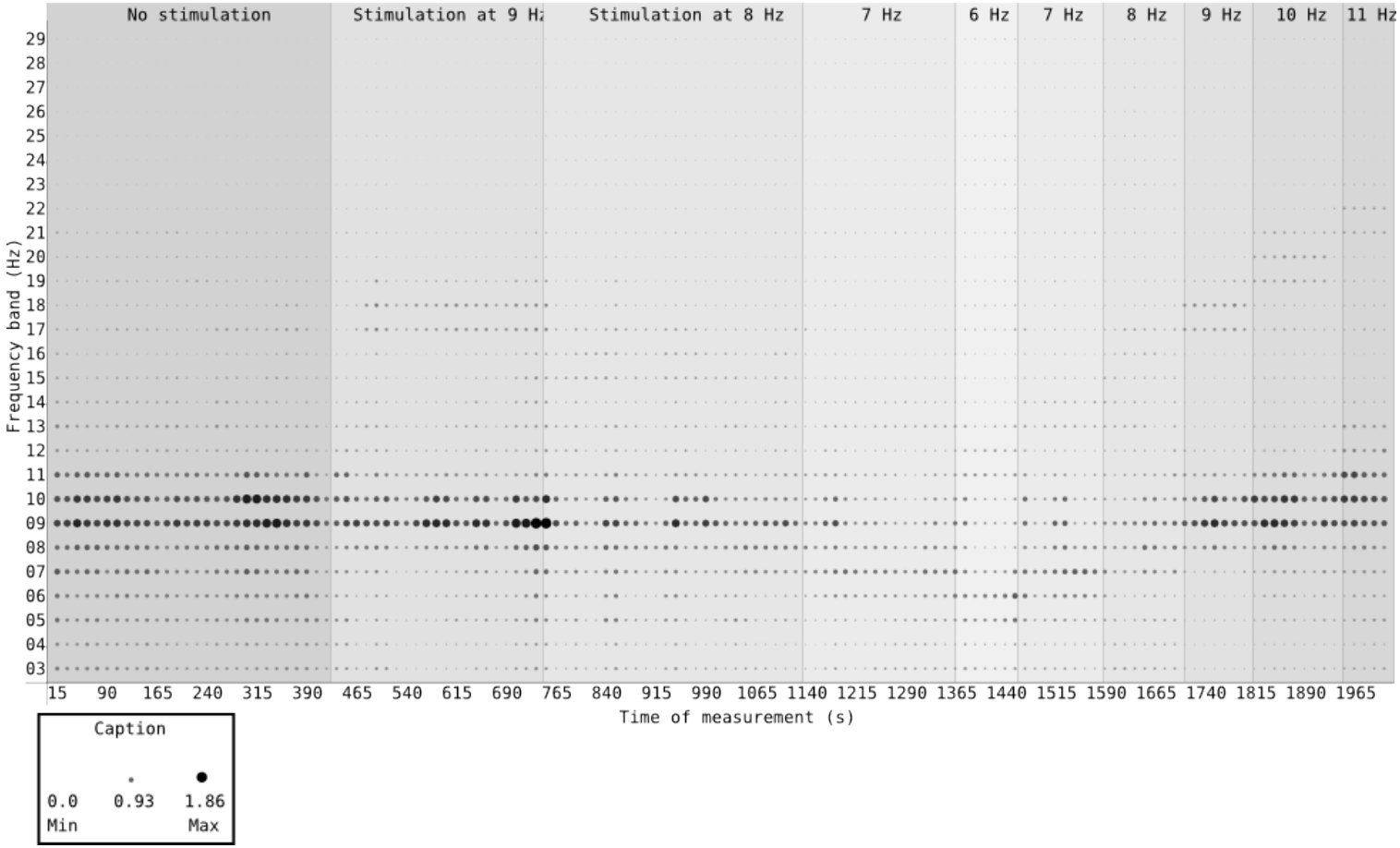


Figure 8: Test result

5 Discussion

Quintessence of results

It has been shown that building an AVSD and EEG can be done with open source software and hardware and the effect of the AVSD on the human brain can be measured with that EEG. Although not statistically proven since not enough tests with subjects where done, there seems to be a correlation between the median value of the power level of each frequency band and the stimulation frequency. An increase of its median power level can be seen with just a very short delay probably due to averaging the values over time. The increase can be seen when the stimulation at that frequency starts and decreases again when the stimulation stops. The control value has been set manually during the tests but results show that it should be feasible to implement a closed-loop control system for brainwave entrainment. The computational power needed for this seems to be so low that the approach described in this paper can be accomplished with reasonable off the shelf hardware. During tests it has been possible to let subjects sleep and wake up again purposely. So it may even be possible to reach other states of consciousness such as concentration, relaxation or meditation as well.

What still has to be done

- For statistically significant statements further systematic testing has to be done, since this work is a proof of concept only and has been tested with too few subjects. Probably also further insights into the whole topic could be gained.
- Actually write an algorithm that that does the feedback loop.
- Although not any subject complained about the brightness of the VSD, brightness control should be implemented.
- Play the audio tones using the microcontroller. There already exists a small extension board for the Arduino called waveshield[29]. Unfortunately it implements only mono sound, so there would be need to modify it.
- Wireless electrodes which do wireless signal transferring so wearability would be increased. Figure 9 shows the schematic for a ultrashort wave sender. [16, p. 133]
- The EEG signal analysis could be done without a PC by a microcontroller only but the analysis process needs to be optimized a lot to achieve that.

[29] http://www.ladyada.net/make/waveshield/

- Some optimization can be done within BrainBay if you use "Magnitude-elements" instead of band-pass filter and FFT.
- It may be more pleasant for subjects if sinusoidal instead of rectangular waveform for the LEDs are implemented.
- Experiment with the colors of the light. See if subjects entrain better to some frequencies with certain colors.
- Test the effect of frequency patterns instead of single frequencies. Therefor multiple LEDs could be used.

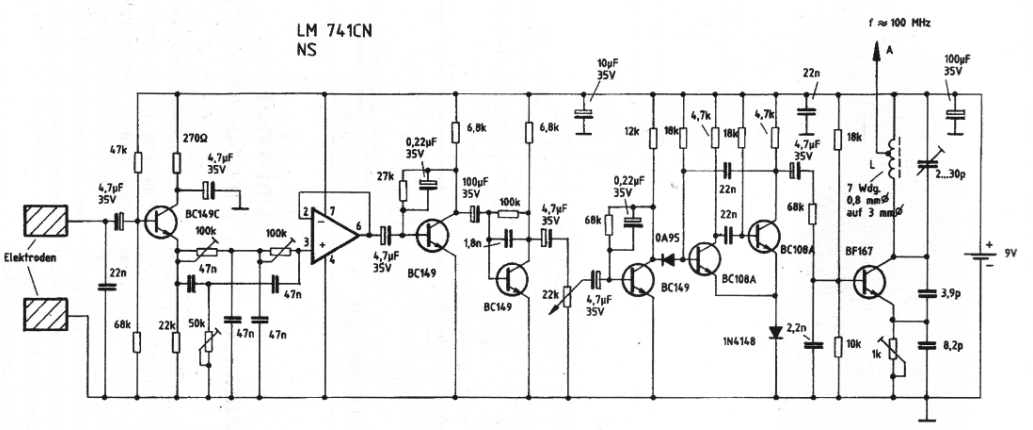


Figure 9: Schematic for ultrashort wave sender

So concluding it seems feasible to build a device by which a human can automatically control his state of consciousness with light pulses and audio tones whose frequency is based on the current dominant brainwave frequency.

6 References

[1] Arduino - HomePage. http://arduino.cc/, December 2009.

[2] Audacity: Free Audio Editor and Recorder. http://audacity.sourceforge.net/, December 2009.

[3] Hans Berger. Über das Elektroenkephalogramm des Menschen. *Archiv für Psychiatrie und Nervenkrankheiten*, 1929.

[4] Thomas Budzynski. The Clinical Guide to Sound and Light. http://www.stanford.edu/group/brainwaves/2006/theclinicalguidetosoundandlight.pdf, 2006.

[5] Sager Rebecca Clayton Martin and Will Udo. In Time with the Music: The Concept of Entrainment and its Significance for Ethnomusicology. *European Meetings in Ethnomusicology, 11 pp. 3–142*, 2005.

[6] Dan Griffiths, Nelo, Jim Peters, Andreas Robinson, Jack Spaar and Yaniv Vilnai. Building the ModularEEG. http://openeeg.sourceforge.net/doc/modeeg/modeeg_building.html, December 2009.

[7] EEG Info Inc. & BEE Medic GmbH. Impedance measurement. http://www.eeginfo.ch/en/neurofeedback/what-is-neurofeedback/how-does-it-work/impedance-measurement.html, December 2009.

[8] EEG Info. Sintered Silver/Silver Chloride Electrodes: Care and Use Instructions. http://www.eeginfo.ch/fileadmin/filemanager/files/Manuals/sintered-electrodes.pdf, July 2009.

[9] Jon Alan Frederick. *EEG Coherence and Amplitude Effects of Rhythmic Auditory and Visual Stimulation*. PhD thesis, University of Tennessee, 2001.

[10] Melinda Cemira Maxfield. *Effects of rhythmic drumming on EEG and subjective experience*. PhD thesis, Institute of Transpersonal Psychology, 1990.

[11] Ian McCulloch. EEG Electrode Impedance Tester. http://www.flickr.com/photos/ianmc333/458904528/, October 2009.

[12] Michal Teplan. Fundamentals of EEG measurement. *MEASUREMENT SCIENCE REVIEW, Volume 2, Section 2*, 2002.

[13] Mark Mumenthaler and Heinrich Mattle. *Fundamentals of Neurology*. Thieme, 1st edition, 2006.

[14] Ernst Niedermeyer and Fernando Lopes Da Silva. *Electroencephalography: Basic Principles, Clinical Applications and Related Fields*. Williams & Wilkins, 4th edition, 2005.

[15] Markus Schubert. Hierarchische Clusterverfahren und deren Anwendung in der EEG-Datenanalyse. Master's thesis, Hochschule für Technik und Wirtschaft Dresden (FH), 2006.

[16] Günter Wahl. *Minispione Schaltungstechnik*. Franzis Verlag, 1st edition, 2006.

[17] Wikipedia, The Free Encyclopedia. 10-20 system (eeg). http://en.wikipedia.org/w/index.php?title=10-20_system_(EEG)&oldid=261403000, December 2009.

[18] Wikipedia, The Free Encyclopedia. Brainwave entrainment. http://en.wikipedia.org/w/index.php?title=Brainwave_entrainment&oldid=329191396, December 2009.

[19] Wikipedia, The Free Encyclopedia. Evoked potential. http://en.wikipedia.org/w/index.php?title=Evoked_potential&oldid=320886306, December 2009.

[20] Wikipedia, The Free Encyclopedia. Wine_(software). http://en.wikipedia.org/w/index.php?title=Wine_(software)&oldid=332106865, December 2009.

[21] Wikipedia, The Free Encyclopedia. Binaural beats. http://en.wikipedia.org/w/index.php?title=Binaural_beats&oldid=347452432, March 2010.

[22] Stephan Zschocke. *Klinische Elektroenzephalographie*. Springer, 2nd edition, 2002.

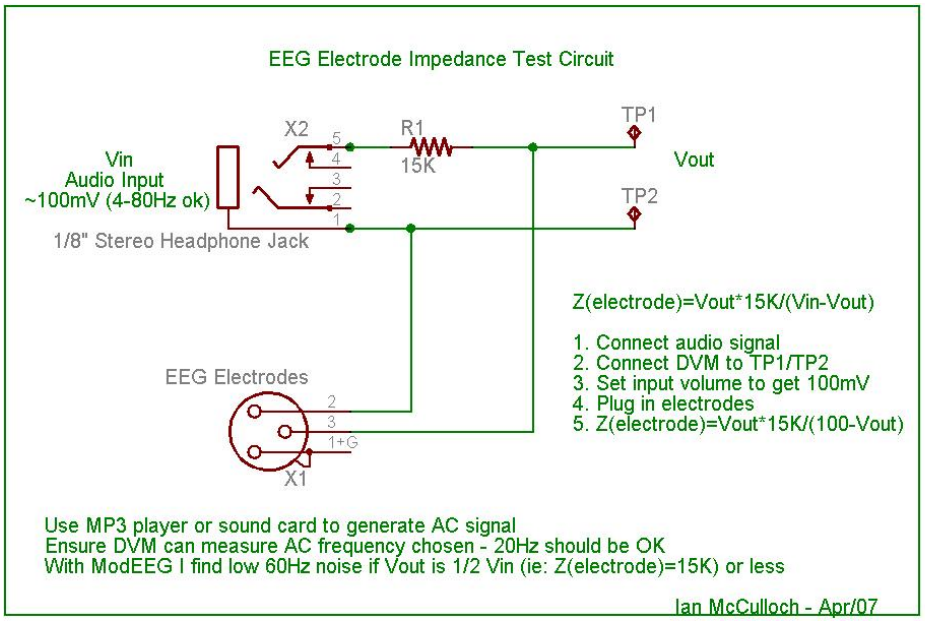
A Appendix

A.1 Instructions for using Ag/AgCl-electrodes

Correct application and cleaning is essential in achieving quality performance and long life for Ag/AgCl electrodes: [8]

- Clean the electrodes immediately after use
- Only use (mild) saline solution or distilled/deionized water for cleaning (no tap water or soap!)
- Do not let the paste dry
- Do not let the electrodes get in touch with metal
- If the electrodes are not being used for several weeks, put them into saline solution for at least three hours
- The skin should be clean and dry where the electrodes will be attached
- Use sufficient amount of paste before use

A.2 EEG electrode impedance tester



"*A very simple impedance tester using a single resistor and an MP3 player or laptop PC sound card for the signal source. Believe it or not, I've found that most of these types of audio sources can pump out signals much lower than you can hear in your headphones...my MP3 player goes right down to 4Hz (a cheap Sansa m240) with not too much drop off. Later I found the electrode impedance between 4Hz and 60Hz didn't vary much more than 5%-10%...so a 30 Hz signal would probably be fine if need be. I created a bunch of 30 second sine wave signals at different frequencies using audacity which I continuously loop on my player when I'm testing.*
I used my DVM to measure the AC voltage between ground and the electrode side of the 15K resistor. Using resistor divider math one can figure out the resistance of the electrode/scalp combination. I made a graph I could reference to determine the electrode impedance but I got to the point where if saw anything less than half my input voltage (meaning effectively 15K impedance) that it was good enough as at that value I tended to see fairly low 60Hz noise in the EEG signal. Nowadays I just use the amount of EEG 60Hz noise to determine how good my electrode connections are to my scalp. But when I'm performing an assessment I pull out my impedance 'meter' to try and be a bit more consistent between the sites for whatever that's worth.
Of course the DVM needs to be able to measure whatever frequency you're driving. My mid-range Radio Shack DVM accurately measures a sine RMS value down to 4Hz (verified by an oscilloscope) which surprised me...but I usually just use 20Hz for my impedance check and I would think most DVM's could handle that.
IMPORTANT - please do not hook this up to a sound card from a PC plugged into wall power...remember you're connecting those electrodes to your head!"
[11]

A.3 "mindparser.py"

The script "mindparser.py" is attached to this Portable Document Format (PDF) file and listed below.

```
1  #!/usr/bin/python
2  # -*- coding:utf8 -*-
3
4  # Name:          mindparser.py
5  # Purpose:       read eeg data parsed by brainbay and calculate
6  #                frequency band power level ratios
7  # Version:       0.0.2
8  # Date:          15.04.2010
9  # License:       GPLv2
10 # Author:        Robert Annessi <robert@annessi.at>
11
12
13 import sys
```

```
14 from time import sleep
15 import select
16 import tty
17 import termios
18 import os
19 from datetime import datetime
20
21
22
23 # frequency band mapping as it appears in the input file
24 freq_map = ['02','03','04','05','06','07','08','09','10','11','12',
25             '13','14','15','16','17','18','19','20','21','22','23',
26             '24','25','26','27','28','29','50']
27 number_of_channels = 2
28 signal_rate = 256 # rate of data received from the eeg
29 measure_len = signal_rate*30 # seconds which data should be stored
30 output_len = measure_len/2 # how often should the median values be printed
31
32
33 if len(sys.argv) < 2 or len(sys.argv) > 3:
34     print "Usage: "+sys.argv[0]+" filename [output_directory]"
35     sys.exit(1)
36
37 # init frequency counter "3d array" (time, channel, frequency_map)
38 freq_cnt = [ [[0]*len(freq_map) for _ in [0]*number_of_channels]
39              for _ in [0]*measure_len ]
40
41 col_cnt = 0 # column counter
42 row_cnt = 0 # row counter
43 f = open(sys.argv[1],'r')
44 reread = 0
45 cur_time = 0.0
46
47
48 # open files for output
49 if len(sys.argv) > 2:
50     f_out = [0]*number_of_channels
51     i = 0
52     for element in f_out:
53         f_out[i] = open(sys.argv[2]+"/ch"+str(i+1)+".txt",'w')
54         i += 1
55
56
57 # for non-blocking read from stdin
58 old_settings = termios.tcgetattr(sys.stdin)
59 tty.setcbreak(sys.stdin.fileno())
60
61
62
63 # print status output
64 print "\nStart: "+str(datetime.now())+"  Frequencies: "+str(len(freq_map))+\
65       "  Channels: "+str(number_of_channels)+"  Rate: "+str(signal_rate)+\
66       "Hz  Measure: "+str(measure_len/signal_rate)+"s  Output: "+\
67       str(output_len/signal_rate)+"s\n"
68
69 # Return the median of the list of numbers.
70 def median(numbers):
71     n = len(numbers)
72     numbers.sort()
73
74     if n & 1: # There is an odd number of elements
75         return nunmbers[n/2]
```

```
76      else:
77          return (numbers[n/2-1] + numbers[n/2]) /2
78
79
80
81  def calc_dom_freq():
82      global cur_time
83      cur_time += float(row_cnt/signal_rate)
84      print str(cur_time)+"s"
85      assessment = [[0]*len(freq_map) for _ in [0]*number_of_channels]
86      assessment_calc = [[0]*(len(freq_map)-1)
87                         for _ in [0]*number_of_channels]
88      print "␣",
89      for i in range(0, len(freq_map)):
90          print "%02iHz" % int(freq_map[i]),
91          for k in range(0, number_of_channels):
92              tmp = [0]*measure_len
93              for j in range(0, measure_len):
94                  tmp[j] = freq_cnt[j][k][i]
95              assessment[k][i] = median(tmp)
96              # copy assessment array without 50hz value
97              # as it would always be the maximum
98              if i < len(freq_map)-1:
99                  assessment_calc[k][i] = assessment[k][i]
100             # file output
101             if len(sys.argv) > 2:
102                 f_out[k].write(str(assessment[k][i])+"␣")
103
104     # file output
105     if len(sys.argv) > 2:
106       for i in range(0, number_of_channels):
107           f_out[i].write("\n")
108     print ""
109
110     # readable debug output for calculated median array
111     for row in assessment:
112         print "[",
113         for element in row:
114             print "%4.2f" % round(element, 2),
115         print "]"
116
117     dominant_frequencies = [0]*number_of_channels
118     print "Dominant␣frequencies:",
119     for i in range(0, number_of_channels):
120         if assessment_calc[i].count(max(assessment_calc[i])) == 1:
121             cur_dom_freq = freq_map[assessment_calc[i].index(max(
                    assessment_calc[i]))]
122             dominant_frequencies[i] = cur_dom_freq
123             # print dominant frequency value + value to 50hz power level
124             print cur_dom_freq+"(%3.2f)" % float(max(assessment_calc[i])/
                    assessment[i][len(freq_map)-1]),
125         else:
126             dominant_frequencies[i] = 0
127             print "NA",
128     print ""
129     print ""
130     return dominant_frequencies
131
132
133
134 try:
135     while 1:
```

```
136         # calculate result if ctrl+d is pressed
137         if select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], []):
138             if sys.stdin.read(1) == '\x04':
139                 calc_dom_freq()
140 
141         line = f.readline()
142 
143         # eof reached, wait for new data
144         if not line:
145             sleep(0.1)
146             continue
147         else:
148             # if the current line has not been written completely yet
149             # wait and reread
150             if ord(line[len(line)-1]) != 10:
151                 f.seek(f.tell()-len(line))
152                 sleep(0.1)
153                 continue
154 
155             newdata = [[0]*len(freq_map) for _ in [0]*number_of_channels]
156             for element in line.split(","):
157                 # FIXME: this should not be necessary, but sometimes an
158                 # element in the last line is corrupted but ok when the
159                 # line is reread
160                 try:
161                     # cast to float since there is a space in front
162                     element = float(element)
163                 except:
164                     # last element hasnt been written completely:
165                     #  reset vars and reread
166                     f.seek(f.tell()-len(line))
167                     col_cnt = 0
168                     sleep(0.1)
169                     reread = 1
170                     break
171 
172                 cur_ch = int(col_cnt/len(freq_map))
173                 newdata[cur_ch][col_cnt-cur_ch*len(freq_map)] = element
174                 col_cnt += 1
175 
176             if reread == 1:
177                 reread = 0
178                 continue
179 
180             row_cnt += 1
181 
182             # delete oldest and add new data
183             freq_cnt.pop(0)
184             freq_cnt.append(newdata)
185 
186             # output after measure limit
187             if row_cnt >= output_len:
188                 dominant_frequencies = calc_dom_freq()
189                 row_cnt = 0
190 
191             col_cnt = 0
192 
193 except KeyboardInterrupt:
194     print "\nQuitting..\n"
195     f.close()
196     termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_settings)
197 
```

```
198     # close output files
199     if len(sys.argv) > 2:
200         i = 0
201         for element in f_out:
202             f_out[i].close()
203             i += 1
204 
205     sys.exit(0)
```

A.4 “mindcommander.py”

The script “mindcommander.py” is attached to this PDF file and listed below.

```
1  #!/usr/bin/python
2  # -*- coding: utf8 -*-
3  
4  # Name:          mindcommander.py
5  # Purpose:       send commands to a mindmachine and play sounds
6  # Version:       0.0.1
7  # Date:          29.08.2009
8  # License:       GPLv2
9  # Author:        Robert Annessi <robert@annessi.at>
10 
11 
12 import sys
13 import serial
14 import pygame
15 from time import sleep
16 import select
17 import tty
18 import termios
19 from datetime import datetime
20 
21 
22 
23 if len(sys.argv) != 2:
24     print "Usage: "+sys.argv[0]+" [serialdevice]"
25     sys.exit(1)
26 
27 
28 # init serial port
29 ser = serial.Serial(port=sys.argv[1], timeout=1)
30 
31 # init sound
32 pygame.mixer.init(44100,-16,2,1024)
33 leftS = pygame.mixer.Sound("../sounds/200.wav")
34 leftCh = pygame.mixer.Channel(0)
35 rightCh = pygame.mixer.Channel(1)
36 
37 
38 # init vars
39 feedback = 0
40 input = "%02d" % 00
41 
42 start = datetime.now()
43 print "\nStart: "+str(start)+"\n"
44 
```

```
try:
    while True:
        # read from stdin (non-blocking)
        if select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], []):
            line = sys.stdin.readline()
            # values between 2 and 29
            if line:
                try:
                    input = int(line)
                    if input >= 2 and input <= 29:
                        input = "%02d" % input
                        feedback = 1
                    else:
                        input = "%02d" % 00
                        feedback = 0
                except:
                    continue

        # change sound
        if feedback == 1:
            print datetime.now()-start
            print "Setting sound to "+input+"Hz"
            rightS = pygame.mixer.Sound("../sounds/2"+input+".wav")
            leftCh.play(leftS, -1)
            leftCh.set_volume(1, 0)
            rightCh.play(rightS, -1)
            rightCh.set_volume(0, 1)

        # mindmachine communication..
        if feedback == 1:
            print "Sending \""+input+"\" to mindmachine:",
            sys.stdout.flush()
            ser.write(input)
            output = ser.read(2)
            if output != input:
                print "ERROR ("+output+")!\n"
            else:
                print "OK!\n"
            feedback = 0


        sleep(0.1)



except KeyboardInterrupt:
    print "SIGINT received: quitting.."
    ser.close()
    pygame.mixer.quit()
    sys.exit(0)
```

A.5 "mindcontroller.c"

The firmware "mindcontroller.c" is attached to this PDF file and listed below.

```
/*
 * Name:          mindcontroller.c
```

```
3   * Purpose:       alternate blinking of leds at certain frequency
4   *                specified through USART
5   * Version:       0.0.1
6   * Date:          11.09.2009
7   * License:       GPLv2
8   * Author:        Robert Annessi <robert@annessi.at>
9   */
10
11
12  /*
13   * Waits for data through USART (9600 8N1).
14   * Expects numbers between 02 and 29 (=frequency in Hz)
15   * LEDs blink at that frequency
16   */
17
18
19  #include <avr/io.h>
20  #include <avr/interrupt.h>
21  #include <avr/sleep.h>
22
23
24  #define LED1_PIN 1
25  #define LED2_PIN 2
26  #define TIMER1_PRESCALER 64
27
28
29  volatile uint8_t digit1;
30  volatile uint8_t digit2;
31
32
33  // set timer1 interrupt to specified frequency
34  void updateTimer(uint8_t frequency) {
35          // turn led1 on and led2 off
36          PORTB |= (1<<LED1_PIN);
37          PORTB &= ~(1<<LED2_PIN);
38
39          // reset counter
40          TCNT1 = 0x00;
41
42          // set top value
43          OCR1A = F_CPU/(2*TIMER1_PRESCALER*frequency)-1;
44
45          // enable output compare interrupt
46          TIMSK1 |= (1<<OCIE1A);
47  }
48
49
50  void setup(void) {
51          // set (leds) to output
52          DDRB |= (1<<LED1_PIN) | (1<<LED2_PIN);
53
54          // init timer
55          // enable ctc mode
56          TCCR1B = (TCCR1B|(1<<WGM12))&~((1<<WGM13)|(1<<WGM11)|(1<<WGM10));
57          // prescaler 64 (greatest scale with min frequency 2hz possible:
58          // 16000000/(2*64*2)-1 = 62499)
59          TCCR1B = (TCCR1B|(1<<CS11)|(1<<CS10))&~(1<<CS12);
60
61          // init usart
62          // baudrate 9600
63          UBRR0L = 103;
64          // enable receiver, transmitter and receive complete interrupt
```

```
65          UCSR0B |= ((1<<RXCIE0) | (1<<RXEN0) | (1<<TXEN0));
66          // 8 Bit character size (UCSZ0-UCSZ2)
67          // No parity (UPM00-UPM01)
68          // 1 stop bit (USBS)
69          // Asynchronus (UMSEL00-UMSEL01, UCPOL)
70          UCSR0B &= ~((1<<UCSZ02));
71          UCSR0C = (0<<UMSEL01) | (0<<UMSEL00) | (0<<UPM01) | (0<<UPM00) \
72                   | (0<<USBS0) | (1<<UCSZ01) | (1<<UCSZ00) | (0<<UCPOL0);
73
74          // configure idle mode
75          set_sleep_mode(SLEEP_MODE_IDLE);
76
77          // enable global interrupts
78          sei();
79  }
80
81
82  int main(void) {
83          setup();
84
85          for (;;) {
86                  if (digit1 != 0 && digit2 != 0) {
87                          // Wait for empty transmit buffer and send data
88                          while (!(UCSR0A & (1 << UDRE0)));
89                          UDR0 = digit1;
90                          while (!(UCSR0A & (1 << UDRE0)));
91                          UDR0 = digit2;
92
93                          // setting timer to specified frequency
94                          updateTimer((digit1-0x30)*10+(digit2-0x30));
95
96                          // reset data
97                          digit1 = digit2 = 0;
98                  }
99                  sleep_enable();
100         }
101 }
102
103
104 // send received char back on usart receive complete interrupt
105 ISR(USART_RX_vect) {
106         if (digit1 == 0)
107                 digit1 = UDR0;
108         else if (digit2 == 0)
109                 digit2 = UDR0;
110 }
111
112
113 // toggle leds on timer1 compare match interrupt
114 ISR(TIMER1_COMPA_vect) {
115         PORTB ^= (1<<LED1_PIN);
116         PORTB ^= (1<<LED2_PIN);
117 }
```

A.6 "mindplot.py"

The script "mindplot.py" is attached to this PDF file and listed below.

```
#!/usr/bin/python
# -*- coding:utf8 -*-

# Name:          mindplot.py
# Purpose:       create svg plot of mindparser data
# Version:       0.0.1
# Date:          03.05.2010
# License:       GPLv2
# Author:        Robert Annessi <robert@annessi.at>


import sys
import math
from pysvg.builders import *



freq_map = ['03','04','05','06','07','08','09','10','11','12','13',
            '14','15','16','17','18','19','20','21','22','23','24',
            '25','26','27','28','29'] # frequency mapping
bars = [ ['0','0'], ['426','9'], ['746','8'], ['1135','7'],
         ['1364','6'], ['1459','7'], ['1588','8'], ['1710','9'],
         ['1812','10'], ['1947','11'] ] # stimulation frequencies + length
linespace = 17 # line spacing in pixels
distance = 7 # distance between circles
ylabeldist = 1 # distance between y axis labels
key_min = "Min"  # key label for minimum
key_max = "Max"  # key label for maximum
frame_width = 0.3 # width of frame lines
frame_space = 6 # space between frame and circles
bar_offset = 190 # color offset of stimuli bars
scale_factor = 2 # circle scale value
min_gray = 255 # minimal gray value of circles
max_gray = 50 # maximum gray value of circles
max_gray_bar = 210 # maximum gray value of stimuli bars
min_gray_bar = 240 # minimum gray value of stimuli bars



# svg init
s = svg()
myStyle = StyleBuilder()
myStyle.setFontFamily(fontfamily="Monospace")
myStyle.setFontSize('0.8em')
font_width = 6.2 # roughed font width
oh = ShapeBuilder()


# check if filename is present
if len(sys.argv) != 4:
    print "Usage: "+sys.argv[0]+" filename seconds seconds_offset"
    sys.exit(1)

secs = int(sys.argv[2])
secs_off = int(sys.argv[3])



# return hex representation of decimal number
def dec2hex(n):
  limit = 255
  if n > limit:
```

```
63      sys.stderr.write("ERROR: dec2hex(): number("+str(n)+
64                       ") too large(>"+str(limit)+")!")
65      sys.exit(2)
66
67    return "%02X" % n
68
69
70  # calculate color value for given float
71  def calcColor(n):
72    diff = min_gray-max_gray
73    ratio = n/(greatest/diff)
74    ratio = int(round(ratio))
75    ratio += max_gray
76    ratio = 255-ratio
77    return str(dec2hex(ratio))
78
79
80  # get longest string size of frequency map items
81  freq_string_size = 0
82  for i in range(0, len(freq_map)):
83    if len(freq_map[i]) > freq_string_size:
84      freq_string_size = len(freq_map[i])
85  x_val_offset = (freq_string_size+1.5)*font_width+frame_space
86
87
88  # total measurement values
89  measure_vals = 0
90  f = open(sys.argv[1], 'r')
91  while True:
92    line = f.readline()
93    if not line:
94      break
95    measure_vals += 1
96
97
98  # calculate y offset
99  y_val_offset = font_width+frame_space
100
101
102 # calc stimulation frequency band range
103 maxbar = 0
104 minbar = sys.maxint
105 for i in range(0, len(bars)):
106   val = float(bars[i][1])
107   if val == 0:
108     continue
109   if val < minbar:
110     minbar = val
111   elif val > maxbar:
112     maxbar = val
113
114
115 ## create sorted list of uniq stimulatino frequencies
116 # create list out of frequencies
117 bar_rank = []
118 for i in range(0, len(bars)):
119   val = int(bars[i][1])
120   if val != 0:
121     bar_rank.append(val)
122 # remove duplicates from list
123 bar_uniq = []
124 for x in bar_rank:
```

```
125     if x not in bar_uniq:
126       bar_uniq.append(x)
127   # sort uniq list
128   bar_uniq.sort()
129
130
131   # create stimuli bars
132   bar_start = 0
133   # stimulationm frequency ratio
134   part = float(min_gray_bar-max_gray_bar)/(maxbar-minbar+1)
135   for i in range(0, len(bars)):
136     # color
137     val = int(bars[i][1])
138     if val == 0:
139       val = dec2hex(max_gray_bar)
140     else:
141       cnt = bar_uniq.index(val)
142       val = min_gray_bar-cnt*part
143       val = int(round(val))
144       val = dec2hex(val)
145
146     # start
147     bar_start = int(bars[i][0])
148     bar_start = float(bar_start)/secs*distance
149     bar_start = int(round(bar_start))
150
151     # width
152     if i != len(bars)-1:
153       bar_width = float(bars[i+1][0])-float(bars[i][0])
154       bar_width = bar_width/secs*distance
155       bar_width = int(round(bar_width))
156     else:
157       bar_width = measure_vals*distance-bar_start
158
159     # draw bar
160     linestart = x_val_offset+bar_start
161     lineend = len(freq_map)*linespace+frame_space+y_val_offset
162     s.addElement(oh.createRect(linestart, 0, bar_width,
163                  lineend, strokewidth=0, fill="#"+val+val+val))
164     # label bars
165     val = int(bars[i][1])
166     if val == 0:
167       t = "No stimulation"
168     else:
169       t = "Stimulation at "+str(val)+" Hz"
170     if len(t)*font_width >= bar_width:
171       t = str(val)+" Hz"
172     if len(t)*font_width >= bar_width:
173       t = str(val)
174
175     # add label
176     val = float(x_val_offset)
177     val += bar_start
178     val += float(bar_width)/2
179     val -= float(len(t))*font_width/2
180     t = text(t, val, y_val_offset)
181     t.set_style(myStyle.getStyle())
182     s.addElement(t)
183
184     # add frames
185     s.addElement(oh.createLine(linestart, 0, linestart,
186                  lineend, strokewidth=frame_width, stroke="gray"))
```

```
187
188 s.addElement(oh.createLine(linestart+bar_width, 0, linestart+bar_width,
189             lineend, strokewidth=frame_width, stroke="gray"))
190
191
192 # get greatest element
193 greatest = 0
194 smallest = sys.maxint
195 f.seek(0)
196 while True:
197   line = f.readline()
198   if not line:
199     break
200
201   for element in line.split():
202     element = float(element)
203     if element > greatest:
204       greatest = element
205     elif element < smallest:
206       smallest = element
207
208
209 greatest_real = greatest
210 greatest *= scale_factor
211 smallest_real = smallest
212 smallest *= scale_factor
213
214
215 # read values from file and draw circles
216 f.seek(0)
217 measure_vals = 0
218 length = 0
219 f = open(sys.argv[1], 'r')
220 while True:
221   line = f.readline()
222   if not line:
223     break
224
225   # get number of elements in line once
226   if length == 0:
227     length = len(line.split())
228   # check if number of elements in current line
229   elif len(line.split()) != length:
230     sys.stderr.write("Different␣amount␣of␣elements␣in␣line␣"+
231                      str(measure_vals)+"␣("+str(len(line.split()))+
232                      "␣should␣be␣"+str(length)+")")
233     sys.exit(3)
234
235   i = len(freq_map)
236   for element in line.split():
237     element = float(element)
238     element *= scale_factor
239     # draw circle
240     color = calcColor(element)
241     s.addElement(oh.createCircle(x_val_offset+measure_vals*distance,
242                  i*linespace-linespace/4+y_val_offset, element,
243                  strokewidth=0, fill="#"+color+color+color))
244     i -= 1
245
246   measure_vals += 1
247
248
```

```
# frame lines
s.addElement(oh.createLine(x_val_offset, y_val_offset, x_val_offset,
             (len(freq_map)+2)*linespace,
             strokewidth=frame_width, stroke="black"))
s.addElement(oh.createLine(2*font_width,
             len(freq_map)*linespace+frame_space+y_val_offset,
             x_val_offset+measure_vals*distance,
             len(freq_map)*linespace+frame_space+y_val_offset,
             strokewidth=frame_width, stroke="black"))

# create y axis labels
t = "Frequency band (Hz)"
t = text(t, -(len(freq_map)*linespace+len(t)*distance)/2-y_val_offset/2,
         font_width+2*ylabeldist)
t.set_style(myStyle.getStyle())
th = TransformBuilder()
th.setRotation(270)
t.set_transform(th.getTransform())
s.addElement(t)
for i in range(0, len(freq_map)):
  t = text(freq_map[i], 2*font_width,
           (len(freq_map)-i)*linespace+y_val_offset)
  t.set_style(myStyle.getStyle())
  s.addElement(t)


# frame label x
t = "Time of measurement (s)"
t = text(t, x_val_offset+(measure_vals*distance-len(t)*font_width)/2,
         (len(freq_map)+2)*linespace+y_val_offset)
t.set_style(myStyle.getStyle())
s.addElement(t)



# create x axis label
ypos = (len(freq_map)+1)*linespace+y_val_offset
# number of digits needed for max value + distance
val = measure_vals*secs+secs_off
val = math.log(val, 10)+1
val = int(math.floor(val))
labelspace = val+ylabeldist
labelcounter = int(math.floor((measure_vals*distance)/
                              (labelspace*distance)))
for i in range(0, labelcounter):
  t = str(i*labelspace*secs+secs_off)
  t_xpos = x_val_offset+frame_space
  t_xpos = xpos+i*labelspace*distance-len(t)*font_width/2
  t = text(t, t_xpos, ypos)
  t.set_style(myStyle.getStyle())
  s.addElement(t)


avg = (greatest+smallest)/2
avg_real = (greatest_real+smallest_real)/2
# distance is 2* max_elements_width
key_dist = 2*max(len(str(greatest_real))*font_width,
                 len(str(smallest_real))*font_width,
                 len(str(avg_real))*font_width,
                 len(key_min)*font_width,
                 len(key_max)*font_width,
                 2*greatest)
```

```
311
312 # key frame
313 ypos += linespace
314 frameend = 3*key_dist-2*frame_space
315 s.addElement(oh.createRect(x_val_offset-frame_space,
316                            ypos-frame_space, frameend,
317                            5*linespace+frame_space,
318                            strokewidth=2, fill="#FFFFFF"))
319
320 ypos += linespace/2
321 t = "Caption"
322 t = text(t, (x_val_offset+frameend)/2-len(t)*font_width/2, ypos)
323 t.set_style(myStyle.getStyle())
324 s.addElement(t)
325
326
327 # draw circle1
328 xpos = x_val_offset
329 xpos = xpos+max(len(str(smallest_real))*font_width,
330                 len(key_min)*font_width)/2
331 xpos_start = xpos
332 ypos += 2*linespace
333 color = calcColor(smallest)
334 s.addElement(oh.createCircle(xpos, ypos-smallest, smallest,
335              strokewidth=0, fill="#"+color+color+color))
336
337 # print circle1 val
338 xpos_v = xpos-len(str(smallest_real))*font_width/2
339 ypos_v = ypos+linespace
340 t = text(str(smallest_real), xpos_v, ypos_v)
341 t.set_style(myStyle.getStyle())
342 s.addElement(t)
343
344 # print circle1 text
345 xpos_t = xpos-len(key_min)*font_width/2
346 ypos_t = ypos_v+linespace
347 t = text(key_min, xpos_t, ypos_t)
348 t.set_style(myStyle.getStyle())
349 s.addElement(t)
350
351
352 # draw circle2
353 xpos += key_dist
354 color = calcColor(avg)
355 s.addElement(oh.createCircle(xpos, ypos-avg, avg,
356              strokewidth=0, fill="#"+color+color+color))
357
358 # print circle2 val
359 xpos_v = xpos-len(str(avg_real))*font_width/2
360 t = text(str(avg_real), xpos_v, ypos_v)
361 t.set_style(myStyle.getStyle())
362 s.addElement(t)
363
364
365 # draw circle3
366 xpos += key_dist
367 color = calcColor(greatest)
368 s.addElement(oh.createCircle(xpos, ypos-greatest, greatest,
369              strokewidth=0, fill="#"+color+color+color))
370
371 # print circle3 text
372 xpos_t = xpos-len(key_max)*font_width/2
```

```
373 t = text(key_max, xpos_t, ypos_t)
374 t.set_style(myStyle.getStyle())
375 s.addElement(t)
376
377 # print circle3 val
378 xpos_v = xpos-len(str(greatest_real))*font_width/2
379 t = text(str(greatest_real), xpos_v, ypos_v)
380 t.set_style(myStyle.getStyle())
381 s.addElement(t)
382
383
384 print s.getXML(),
```

A.7 ModularEEG cables

The ModularEEG cable schematics file "cables.sch" is attached to this PDF.

A.8 BrainBay configuration

The BrainBay configuration file "brainbay.con" is attached to this PDF.

A.9 BrainBay patches

Increase maximum values

The patch "BrainBay-1.4_increase_max_values.patch" increases the maximum possible connections, ports and objects in BrainBay and is attached to this PDF.

Remove FFT drawings

The patch "BrainBay-1.4_remove-fft-drawing.patch" is attached to this PDF and removes the FFT drawings from BrainBay since they do not work properly under Linux.

A.10 ModularEEG firmware

The ModularEEG firmware "modeeg-p2.c" is attached to this PDF.

A.11 EEG data

The test EEG data “measurement.edf” is attached to this PDF as EDF file.