# Wo issn des Hirn? 2.0

## Technical Documentation

Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Fabio Paul Aubele, Lukas Philipp Edlboeck, Michael Hammerl, Felix Kampfer, Florian Klein, Philipp Weber, Timo Wilhelm

Advisor: Prof. Dr.-Ing. Hon. Dr. of ONPU Thorsten Schöler

## Contents

# 1 Introduction

## 1.1 Motivation

In the world of computers are a lot of different ways to interact with a computer. Most of these methods include an input device manipulated by a user's hands. However, there are people who are unable to use their hands, such as sufferers of Locked-in syndrome. The solution is the implementation of an input device that requires no actual physical contact. One way to realize this is via the use of a brain computer interface.
In the project ”Wo issn das Hirn”, we developed a method to control a variety of ”smart home” devices by using the signals of a human brain. The project is a follow up project of the previous semester project.

## 1.2 State of the Art

Because this is a follow up project, there was already research made, and a prototype of a brain computer interface developed. The previous group tried to use brain signals of two different regions of the brain. One was the signals from the motor cortex.
This cortex of the brain, which is located around the middle of the brain, is involved in the planning, controlling and the execution of movement. The signals were recorded and classified.
The other cortex, which was used to record the data, was the visual cortex. This cortex is located at the back region of the brain. All visual input is being processed in this cortex. In the prototype of the previous group, the user was interacted with a screen showing three boxes, which would flicker with a different frequency. These frequencies are then visible in the recorded brain signals. These signals were then analyzed. The analyzed data was then used to control a drone.
For the analysis of the signals, the previous group used the Naive Bayes classifier. This classifier tries to classify the data on the base of the attributes of the data.

# 2 Concept

Our project can be split into three primary areas of focus: The recording, processing and classification of brainwaves, the use of internet-connected devices and services, and the interaction between these two systems. This chapter explains the underlying concepts of these technologies.

## 2.1 Measuring Brain Waves

One of the most important aspects of our project was analysing the brain's reaction to specific stimuli. To this end, we measured electrical activity in the brain (Electroencephalography) via open-source hardware (OpenBCI), used signal processing software to evaluate a certain type of neural response (OpenVIBE, SSVEP) and then tried to determine the original stimulus (Classification).

One of the most important aspects of our project was analysing the brain's reaction to specific stimuli. To this end, we measured electrical activity in the brain (2.1.1 Electroencephalography) while evoking a certain type of neural response (2.1.2 SSVEP). The open-source hardware we used (2.1.3 OpenBCI) was compatible with an open-source signal processing suite (2.1.4 OpenVIBE) and allowed us to easily filter out the relevant signal data. We also compared a variety of decision algorithms to be able to determine the most likely stimulus for the current neural response (2.1.5 Classification).

### 2.1.1 Electroencephalography

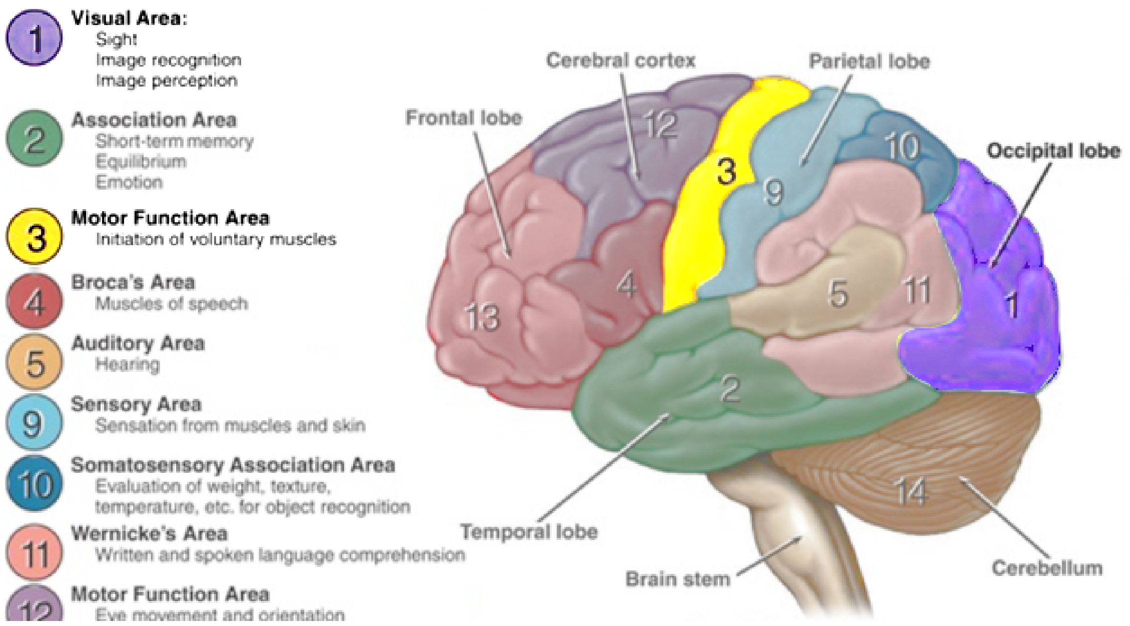


Figure 1: Lateral view of the human brain

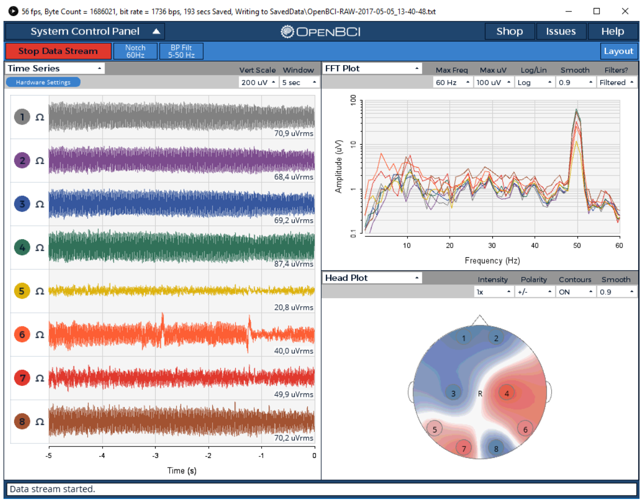
Biopotential telemetry - that is, measuring the electrical potential between two electrodes - allows doctors and researchers to quickly gain insights into neurological, muscular and cardiovascular activity, especially when combined with the signal processing and analytic capabilities of computers. [1] When applied to research endeavours in the field of neurology, electroencephalography (EEG) can be used to measure the effects of internal and external stimuli on the human brain and act as the input to a computer program. This is the case for our brain computer interface project.

The brain is split into different lobes and areas of responsibility. Figure 1 highlights the most relevant areas of the brain for our project in yellow and purple. The yellow area is the motor cortex, which is responsible for controlling the planning, preparation and execution movement. This region was a focus of previous "Wo issn das Hirn?" projects, which employed a methodology called "motor imagery", wherein a subject "imagined" reaching for an object with either his right or left hand. Increased activity in either the left or right half of the motor cortex could then be processed to control the path of a remote control car and/or drone.

### 2.1.2 SSVEP

The purple area of figure 1 shows the occipital lobe, home to the visual cortex and responsible for all visual processing. When a subject focuses on a flickering light source oscillating roughly between 4hz and 40hz, certain parts of the visual cortex are excited with the same frequency. [2] A fourier transform can be used to determine the frequencies for a given signal, provided some additional signal processing is carried out.

### 2.1.3 OpenBCI



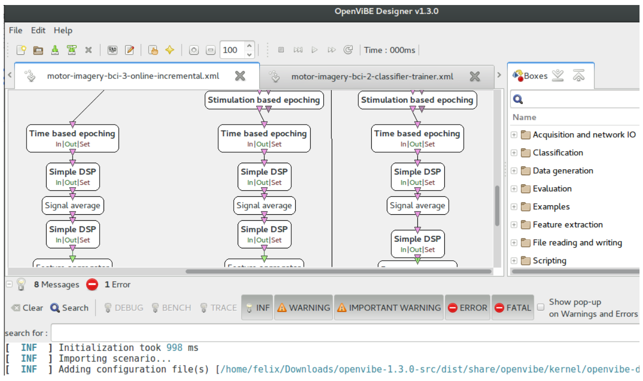
```
56 fps, Byte Count = 1686021, bit rate = 1736 bps, 193 secs Saved, Writing to SavedData\OpenBCI-RAW-2017-05-05_13-40-48.txt
Data stream started.
```

Figure 2: OpenBCI control software for testing electrodes

Our EEG system is comprised of several components: Gold-cup electrodes are placed on the scalp and connect via cable to an electronic amplifier (an OpenBCI Cyton Board [3]), which sends the data wirelessly to an OpenBCI USB dongle plugged into a signal processing computer. The electrodes are kept firmly in place thanks to custom-printed electrode holders and a flexible EEG cap.

The OpenBCI Cyton Board "can be used to sample brain activity (EEG), muscle activity (EMG), and heart activity (ECG)" with data being sampled 250 times per second, according to the manufacturer. The wireless transmission takes places via so-called RFduino modules, and is unfortunately prone to significant interference when people are walking between the sender and receiver. In order to test the electrode connections, OpenBCI ships with an open-source control software (see figure 2) to verify a strong signal before engaging in advanced signal processing. [3]

### 2.1.4 OpenVIBE



```
[   INF   ] Initialization took 998 ms
[   INF   ] Importing scenario...
[   INF   ] Adding configuration file(s) [/home/felix/Downloads/openvibe-1.3.0-src/dist/share/openvibe/kernel/openvibe-d
```

Figure 3: OpenVIBE signal processing suite

OpenVIBE is an French XML-based open-source editor for signal processing. Users use boxes and lines to connect inputs to mathematical operations, visual graphs, custom functions and even external Python scripts in order to facilitate their individual signal processing needs while maintaining a high degree of transparency and comprehensibility.

#### 2.1.5 Classification

We tested different algorithms for classification of the signal. Instead of using the classifiers that come with OpenVibe we used Python-boxes in OpenVibe to integrate our own classifiers. These utilized librarys such as scikit-learn and a combination of Theano, Lasagne and nolearn. This approach has two advantages: Firstly we could controll all variables and every step of the classifiers or use Algorithms that are not included in OpenVibe, such as Convolutional Neural Networks. Additionally the Python librarys we used provided as significant performance boost over the implementations in OpenVibe.

## 2.2 Home Automation Components

In our home automation scenario we use two components, which we want to regulate with brain signals. Those two component get controlled in the app we developed in android.

#### 2.2.1 Spotify

Based on the official Spotify API for android, we implemented a simple music player, which allows us to play/pause a song, and jump to the next or previous song in the play-list. The play-list is hard coded in the app due to missing configuration possibilities from the API. Also the play-list always runs with shuffle and repeat. To use the Spotify feature correctly you need to login on the configuration site of our app. We provided a Spotify account in the GitLab-Wiki. Be careful Spotify premium is a necessity and it could already run out on the account. You need internet connection for this since the app is communicating with the Spotify-servers. If you want to use an other account you first need to register the developed application on this website [4], you can find a precise tutorial for that on this page [5]

#### 2.2.2 Sockets

Our system also uses some WIFI-sockets that can be turned on and off in our app we developed in android. The socket-systems producer is EASY HOME from a german discounter. Those sockets allow us to control the power with a WIFI-connection. To use the sockets you need to enter the IP-Address of the sockets on the configuration site of our app.
In order to establish a connection between the sockets and a local network, you first have to go through a pairing process in the manufacturer's app. This will transmit the network SSID and password to the socket, enabling it to connect to the network. With the connection established the socket listens for a certain hash format via UDP. The composition of this hash is explained in 3.3.4. When it receives a valid command hash, the socket will send back that identical hash as a confirmation. The second socket of each pair is controlled via the 433 Mhz ISM frequency band. The primary socket will forward any hash that contains the associated RF address to the secondary plug, allowing both sockets to be controlled sending suitable hashes to the main device.

## 2.3 Home Automation Control

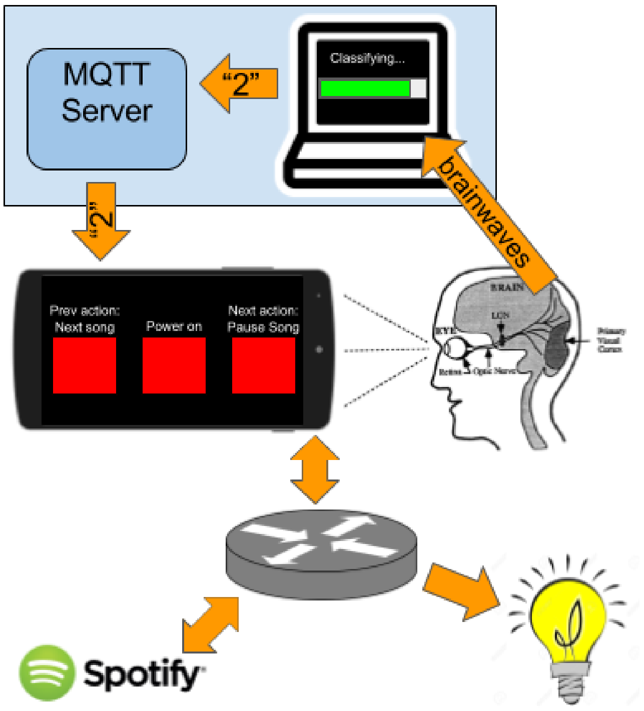


Figure 4: <Interaction between components>

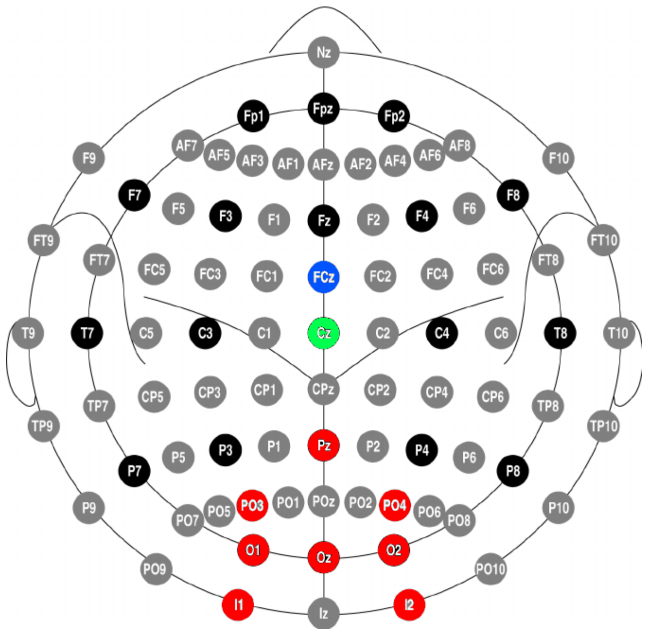
There are two main components that are crucial for the configuration to work. First of all, the user can navigate through our android app by staring at one of the three rectangles. Each of these flicker at a unique, given refresh rate. The outer rectangles are used to navigate between the menu items, while the inner one executes the selected option.
The incoming brain signals are constantly being processed and analysed by the system that is connected to the OpenBCI kit. OpenVIBE and our classifier are responsible for the actual evaluation. As soon there is enough confidence in a certain signal over a given period of time, the result will be sent via MQTT as a simple number, corresponding to the index of the focused rectangle. The associated server is also installed on the same machine.
The android app is connected to that MQTT-Server, listening for any events on the related topic. Once it receives a new signal being detected, it reacts depending on the rectangle the user has looked at. If that is the inner rectangle, the app executes the code that is associated to the action. Otherwise the user interface browses one menu item back/forth. Currently our app contains possibilities to control spotify by using their android API as well as switching the mentioned pair of WIFI-sockets on and off. However, with this scalable architecture, further devices and commands can easily be implemented by editing the android app source code only.

# 3 Implementation

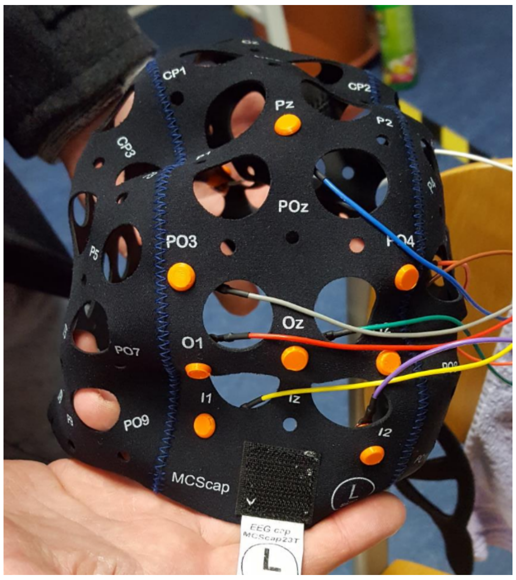
## 3.1 Hardware Setup

Setting up the hardware requires multiple steps: First the electrodes have to be placed on the subjects head and connected to the EEG sensor. After the sonsor is connected to the computer the Android device has to register itself to receive the interpetation of the EEG signal. Finally the remote controlled plugs have to be connected to the Android device so they can be actuated.

### 3.1.1 Electrode Placement


Nz
Fp1 Fpz Fp2
AF7 AF5 AF3 AF1 AFz AF2 AF4 AF6 AF8
F9 F7 F5 F3 F1 Fz F2 F4 F6 F8 F10
FT9 FT7 FC5 FC3 FC1 FCz FC2 FC4 FC6 FT8 FT10
T9 T7 C5 C3 C1 Cz C2 C4 C6 T8 T10
TP9 TP7 CP5 CP3 CP1 CPz CP2 CP4 CP6 TP8 TP10
P9 P7 P5 P3 P1 Pz P2 P4 P6 P8 P10
PO7 PO5 PO3 PO1 POz PO2 PO4 PO6 PO8
PO9 O1 Oz O2 PO10
I1 Iz I2

(a) 10-20 Diagram with relevant positions marked: Blue is bias and green is ground (reference).



(b) Lead Placement on the flexible cap

Figure 5: Electrode placement

In order to be able to conduct experiments in a reproducible manner, we use an exteded version of the so called 10-20 system, which divides the hemisphere of the brain into steps of 10-20%, allowing for the precise placement of electrodes. [6] A flexible EEG cap is used to keep the electrodes in place. When putting the cap on the Cz Position has to be on the center between the ears and between the nasion and the inion. Than the electrodes are fixed in the positions marked in Figure 5a with our custom printed holders. When placing an electode it has to be ensured that a good contact to the scalp and the electrode is present. This is done by moving the hair aside and applying some conductive EEG-paste to the electrode.

### 3.1.2 EEG Sensor Setup

The wires of the electrodes are then connected to the OpenBCI EEG sensor. Since we are using passive electrodes the signal is very prone to interference. The wires should therefore be as short as possible. The FCz electrode is connected to the BIAS pin and the Cz electrode is to the reference (SRB2) pin. The remaining electrodes are assigned to the pins 1N to 8N. The sensor can then be connected to the computer by first plugging the RFduino and then turning on the OpenBCI board. In the OpenBCI GUI a voltage can be seen for each channel. This voltage is measured by letting a small current flow between the electrodes. It corresponds to the resistance between skalp and electrode. The voltage should be reduced to a value smaller than 10 µVrms by adjusting the electrode.

### 3.1.3 Visual Stimuli

The eyes are stimulated by a custon Android application which shows multiple boxes alternating between black and red. The frequencies with wich they are blinking should be dividers of the screen refresh rate and within the range of 3.5 Hz to 75 Hz. When choosing frequencies it is important that the candidates are not multiples of one another. In our experiment we choose the frequencies 7.5 Hz, 10 Hz and 12 Hz.

### 3.1.4 Pulse Sensor Input

Another possible input we began working on is the pulse sensor which is attached to a finger. The device we used combines an LED with and lightsensor and an amplifier to create an analog signal of the pulse. We used an analog-digital converter and a script running on a raspberry pi to convert this analog signal to a concrete heart-rate. This could be used as another input for our home automation devices, for example for choosing the playlist on the music player. Due to time constraints we where not able to integrate the pulse sensor with our BCI application.

### 3.1.5 Controlled Peripherals

In order to control devices we used remote contolled power sockets. These have to be in the same WiFi the Android device is connected with. The app that comes with the sockets is used to transmit the name and password of the network. After a successful connection the IP of the remote controlled plugs has to be determined and entered into the Android application.

## 3.2 Software Realization

In the following section we will take a look at the software used in the classification process.

The main component for building the data processing pipeline was the OpenVibe toolkit for brain computer interfaces and real time neurosciences.

Using the Python machine learning package scikit-learn, we defined the classifiers as data pipeline elements. Scikit-lern is a popular machine learning library for the Python scripting language. It features a variety of classification, regression and clustering algorithms

The Convolutional Neural Network was constructed with Lasagne, a lightweight library to build and train neural networks in Theano, together with the nolearn wrapper. Theano is a numerical computation library for Python, that features deep integration with NumPy and allows for efficient evaluation of multi-dimensional array operations on CPU or GPU.

### 3.2.1 OpenVIBE Designer Scenarios

The main scenario consisted of seven modules:

1. Configuration of the data acquisition.
2. Recording of sample data with set stimulations.
3. Conversion of the recorded data into CSV format.
4. Configuration of the model training process.
5. Training of the statistical classifier model.
6. Testing of the model with separate datasets.
7. Real-time classification of live input data with the trained model.

### 3.2.2 Auxiliary Scripts

**Data format conversion**
We converted the data from the OpenVibe data format into comma-separated-values to change the way the stimulation events are recorded. OpenVibe sets a flag for the next stimulation type and then sends a start and stop event for the corresponding stimulation. To make it easier to work with the stimulation data, we decided to reduce the number of stimulation events from three messages to only a single one that has a timestamp, type and duration for the stimulation. This transformation helped us in analysing and comparing data from different sources with different data formats.

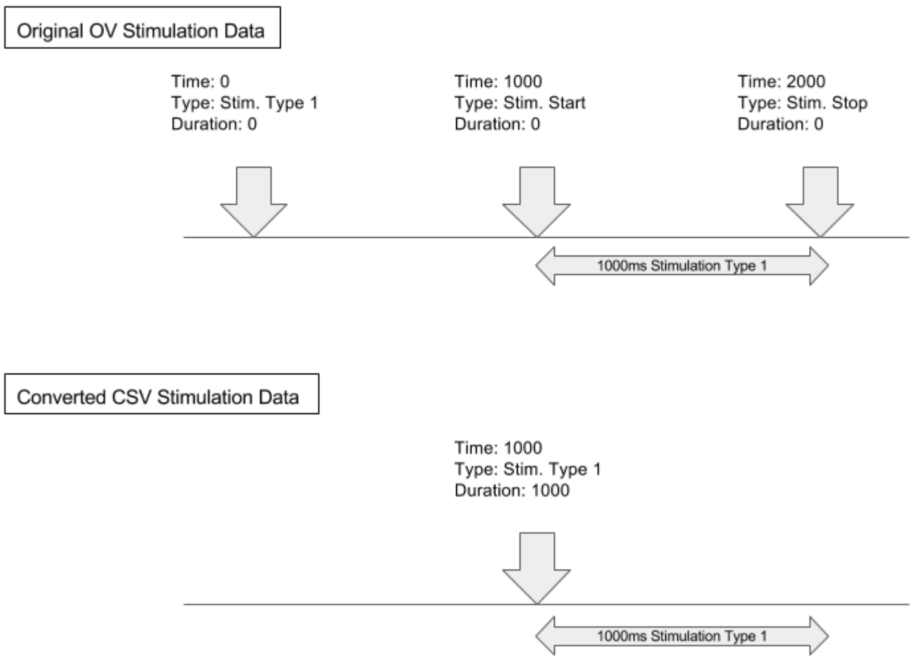


Figure 6: Stimulation Data Formats

**Classifiers**

We focused our research on Gaussian Naive Bayes analysis, Support Vector Machines with a polynominal (degree=3) kernel, and Convolutional Neural Networks. All of those classification methods are supervised learning algorithms, so to train the models we first had to record a set of reference data with known stimulation inputs. For this, we used a modified sample scenario from the OpenVibe SSVEP examples.

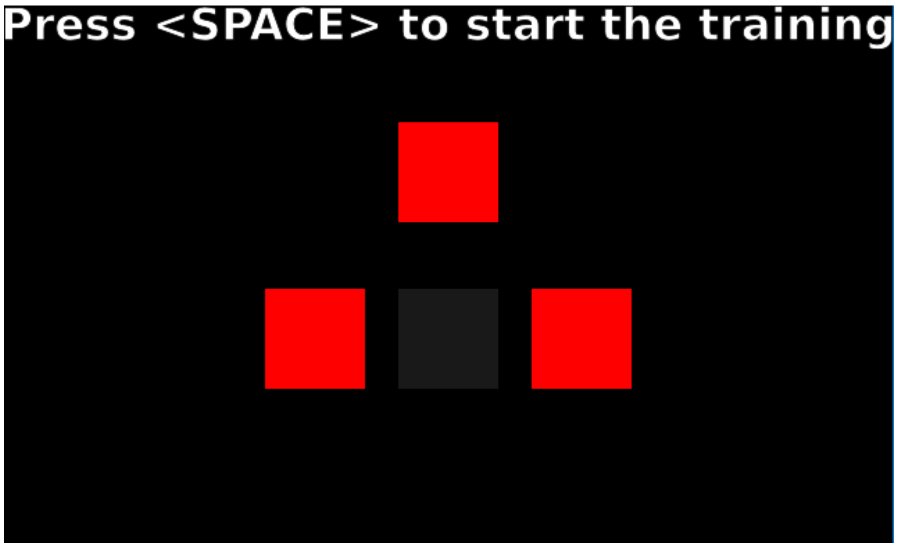


Figure 7: Acquisition UI

**Training Process**

As the data acquisition generates a very uneven number of samples for the different classes, we first had to apply downsampling on the dataset to achieve a uniform distribution and to avoid introducing a bias towards one class for the classifier.

The data format is a 2-dimensional array of 8-Channels with time-series data over 2 seconds with a sampling rate of 250 samples per second. As we apply a fourier transform on the time-series data, we end up with only half the number of samples per channel plus one, with a bucket size of 0.5 Hz. Figure 8 shows the first 60 samples for all channels of the FFT from 0 to 30 Hz. We filter the signals with a Butterworth band pass filter (degree=4) between 3 and 40 Hz to reduce the noise of the power grid at 50 Hz.

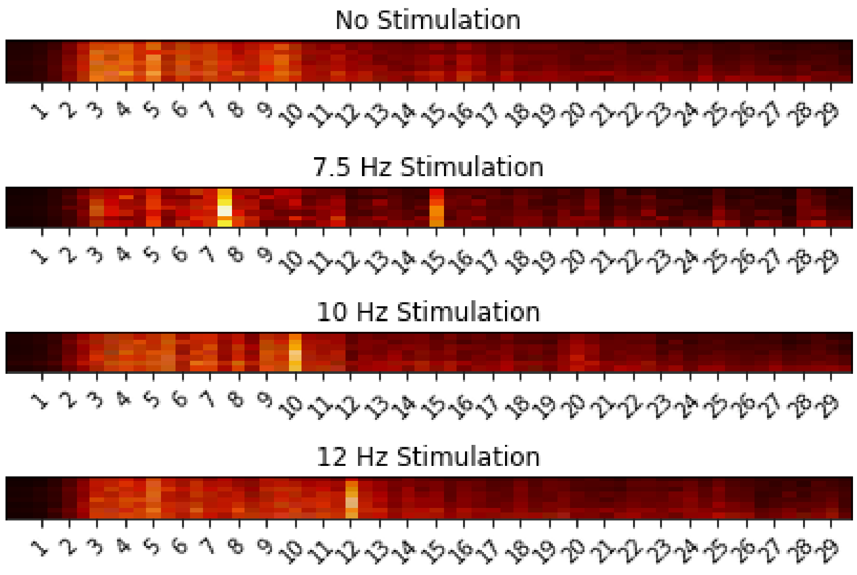


Figure 8: Extract of FFT Signal Data

For training of the Naive Bayes and Support Vector Machines we flattened the 2-dimensional data into a simple 1-dimensional array and using the scikit-learn interface for classifier fitting, trained the different models and saved the resulting python objects to disk.

The Convolutional Neural Network consists of two convolution layer, followed up by two fully connected layers with 128 and 64 neurons respectively. The output layer is a fully connected softmax layer with 4 neurons, representing the Null-state and the three different stimulation.

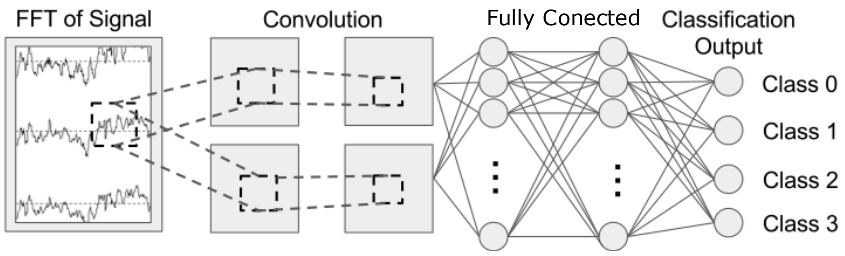


Figure 9: Architecture of the CNN

**Live Prediction**

For live prediction of the signals, the live data is transmitted in real-time to the OpenVibe scenario with the OpenVibe acquisition server. Two-second windows of time-series data, overlapping every 0.2 seconds are fed into the classifier for prediction. To increase the accuracy and avoid outliers during live-prediction, the same stimulation identifier has to be predicted four times within a four second window before a MQTT message is published to the message broker. All subscribed MQTT clients receive the message and trigger the corresponding action.

## 3.3 App Development

### 3.3.1 User Interface

Regarding our project we wanted to use our training results in a proper way to make a good demonstration with it. So we decided to create an application with different features and usage scenarios. The app basically consists of two different activities.
The first activity provides some configuration matters. Here you have to input the necessary data for the MQTT connections (refer 3.3.2) and for the sockets to use everything correctly. Furthermore you can login and logout your Spotify account (refer 3.3.3).
The second activity is the main acitivity and mainly consists of three blinking boxes in red and black. The blinking boxes are realized through ValueAnimator [7]. Here you can simple define two colours and a duration, how fast the animator should switch between those two colours. The design idea behind those three boxes gets explained in 2.3. When you are not logged in with the Spotify account, you only see the actions for the sockets, else you also see the action to play a song. When this action gets triggered the app switches into a playing state and the title of the box 'play a song' now switches to 'pause a song'. Furthermore you now can see the functionality 'next song' and 'previous song'. When you trigger the pause-action, the app switches back to the non-playing state. The boxes flicker in different frequencies. The one on the left with 7.5 Hz, in the middle with 10 Hz and on the right with 12 Hz. When a signal from MQTT gets received or the blinking boxes get clicked, the action titled above will be executed. After a signal from MQTT is received, the specific box is blinking green for a second, to signalize that the action got executed, then it switches back to red.

### 3.3.2 MQTT

The app is reliant on the MQTT signals from the OpenVIBE scenario, which is responsible for evaluation of the brain signals from the user, who is watching the blinking boxes. To use MQTT with the app correctly, you first have to start a MQTT-broker (we used Mosquitto). Then enter the IP-address from the broker and the port specified in the OpenVIBE scenario on the configuration site of the app. When you navigate to the main activity the app tries to establish a connection with the MQTT-broker. While he tries to establish the connection and the broker is not started or the entered data is wrong, it could happen that the app shows a black screen,

till he waits for the MQTT connection to time out. But after some time the app should proceed to show the main activity.
The android app is subscribed on the MQTT-topic 'SSVEP'. The OpenVIBE scenario sends different signals with this topic, reliant on which box the user is looking at. For the left box the OpenVIBE scenario sends 1, for the middle one 2 and for the right one 3. After the signal is received the appropriate box blinks green for a second and its functionality gets executed.

### 3.3.3 Spotify

The developed Android application uses two main functionalities from the Spotify SDK, the first one is to authenticate a user. This is needed so you can play songs and use the SDK correctly. You can login with your Spotify Account on the starting configuration site from the developed app.
The second functionality is to play an audio stream and skip songs as wished. Those actions get triggered from the blinking boxes in the main activity, when you logged in your account. Overall the Spotify SDK and API is really well documented the most important informations can be seen on this site [5]

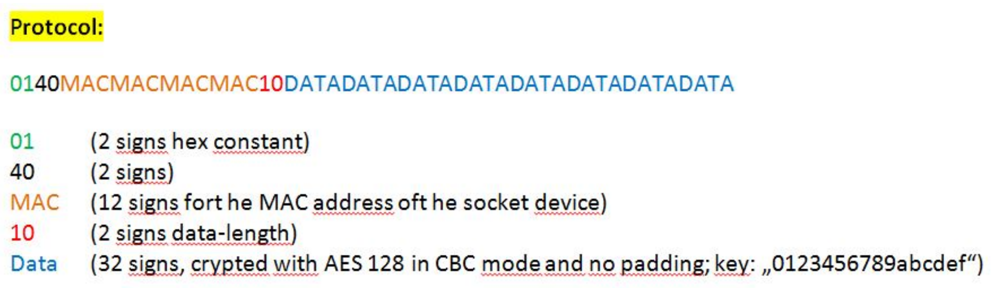
**Authenticate a user**
The documentation from Spotify provides a example on how to set-up the login correctly on this site [5]. Basically after a button got pressed, a login activity provided from Spotify opens, in this activity the user should enter his account credentials and proceed to login his account. After the user finished this step a callback function checks if the login was successful, if so a token gets hand back. With this token it is possible to create a Spotify player instance. The player instance is required to play songs and realize other functionalities for the music stream. In our application the login is in the configuration activity, the token which gets hand back from the login function is passed to the main activity with the blinking boxes, so the music player can get initiate there. We designed it like that, since we only want to play music in the main activity and the login is realised on the configuration site.

**Play an audio stream from Spotify**
Thanks to the music player we created after the login, it is possible to realize certain functionalities. We have the basic function for a music player, consisting of play/pause a song and jump to the next or previous song. All of those functions and how to call them correctly are documented on this site[8]. Basically you just need to provide a play list or a song in the URI-format, which is a link coded in a format Spotify uses to identify each song or play list. Sadly the API provides no possibilities to change the URI in a simple way, so we had to hard-code one play list it in the app. Now all the different music player functions can be realized. In our app the described functionalities are in several methods which get called, after the correct signal from MQTT got send or when the blinking boxes are simply clicked.

### 3.3.4 Hardware control

**How the socket connection works**
In general the app contains some actions inside the blink-boxes, that are responsible for turning the wireless socket device on- and off. To understand, how the implementation of the socket-device basically works, we are going to show you the basic structure of the hash's components, which are sent to the socket device when turning it on and its differences to the mode for turning it off.

The following picture (Figure 10) shows some general components of the hash.



**Protocol:**

0140MACMACMACMAC10DATADATADATADATADATADATADATADATA

| | |
|---|---|
| 01 | (2 signs hex constant) |
| 40 | (2 signs) |
| MAC | (12 signs fort he MAC address oft he socket device) |
| 10 | (2 signs data-length) |
| Data | (32 signs, crypted with AES 128 in CBC mode and no padding; key: „0123456789abcdef") |

Figure 10: Structure of the message sent to the wirelessly controlled socket

As you can see the hash consists out of 50 signs, that are sent to the wireless socket device, every time you are about to switch it on or off. The first to bytes are a static hex constant (green ones) and the following bytes three and four (black ones) are also two constant signs. Then the MAC-address (orange ones) follows. Here you have to set the unique 12 signs of the wireless socket's MAC-address. After these twelve signs two further signs (red ones) are placed. The last part of the hash is the data-part. Here we have 32 signs that are going to be crypted with AES 128 in CBC mode with no padding, before its going to be matched with the previous mentionend part of the hash. It is important to understand that part of the encryption so we are going to explain what the application basically does.

If the blinking-window for turning the socket on is activated the **socketDeviceOn()** function is called. Because you can´t handle network issues and others inside on thread in Android you have to an inner class that extends **AsyncTask**. So that part of the work is handled in background. Now the code tries to open a new **DatagramSocket** and searches for the socket-device using the **ipAddress** it got from the input mask. After it has found the device it creates the packet **sendpacket** which contains the hash, the length of the hash and the ip-address of the socket also as the port where it has to be send. After it has been created it is send 10 times to the socket to make sure, that it arrives there.

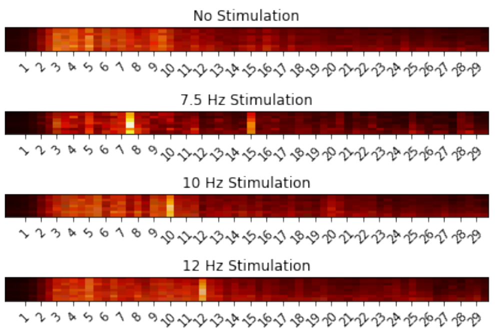
But coming back to the encrytion again. As we said before, the 32 signs long hash has to be encrypted before it can be send. That part of the hash basically consists of 6 Bytes ”00FFFF” of hex code at the beginning followed by 2 bytes of the customer-code which is ”C2” for our sockets. Then another 2 bytes are added to the hash. These two bytes always have the hex-value ”11”. Afterwards the authentication code follows which is ”92DD” also because of customer specifics. And then there is thea another 2 static bytes ”01” before the main state of the socket follows and that part is wether ”0000FFFF” in hex for turning the socket on or ”000000FF” or turning the socket off. At the end of the hash 8 final bytes ”04040404”, are added to complete the hash, so that it can be encrypted now.

The method **HexToBytes** now creates a byte-array out of the string and returns it. That byte-array is then passed to the class **EncryptionAES**, that is responsible for the encryption with AES 128 in Cipher Block Chaining mode. The initialization vector (nonce) for that encryption mode always is ”0123456789abcdef”. So the class encrypts the data of that part of the hash and returns it to the **SocketControl** class where the other uncrypted part of the hash mentioned before (”0140MACMACMACMAC10”)is placed in front of the´now encrypted and returned to the **SocketControl** ready for being. Now the hash can be send to turn the socket on- or off.

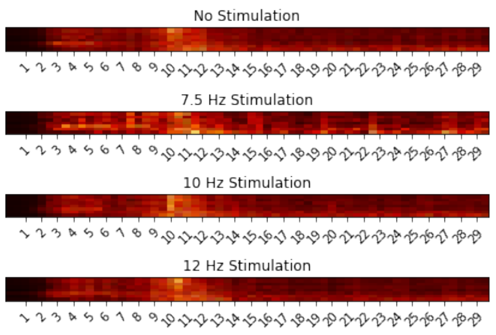
# 4 Conclusion

## 4.1 Evaluation

### 4.1.1 Signal Quality



(a) Signal Quality with reference and bias electrodes on the upper head

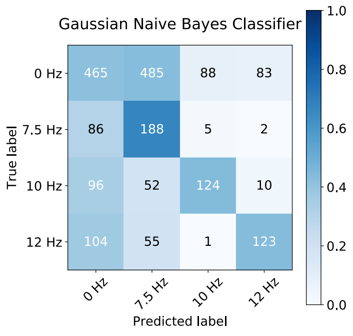


(b) Signal Quality with reference and bias electrodes on the ear lobes

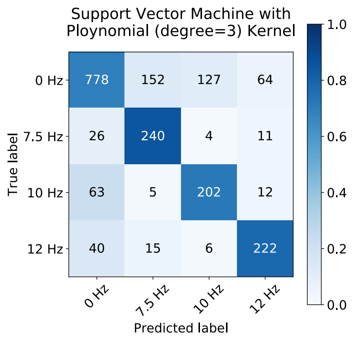
Figure 11: Comparison of Signal Quality

On the right side of Figure 11 is the signal quality of the training data with reference electrodes at the ear lobes. After the visit to the Neurology we changed the position to the upper head and improved the signal quality substantially.

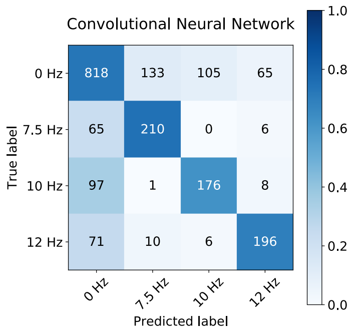
### 4.1.2 Classifier Accuracy



(a) Gaussian Naive Bayes



(b) Support Vector Machine



(c) Convolutional Neural Network

Figure 12: Confusion Matrices

As you can see in Figure 12, the SVM and CNN Classifiers vastly outperformed the Naive Bayes algorithm. While the accuracy of the SVM and CNN is almost identical, the SVM has the advantage of a superior training time. The results could be further improved by a hyper-parameter search. As we did a second filtering over the classified results over a specific time-frame we achieved even better classification for the application.

#### 4.1.3 Application Design

## 4.2 Verdict

Our project group completed both of the goals of our project with great success, stabilizing the signal input in a reproducible fashion and finding new applications for brain computer interfaces. Our hardware improvements (shorter cables, new electrode holders) and software upgrades (a more accurate classifier for SSVEP) will be of great benefit for all future project participants, and may even help the greater scientific community.

We achieved most of our home automation control objectives as well, being able to toggle electricity as well as manipulate a music streaming service. Our initial premise of a ”universal media remote” remains unfulfilled, however, perhaps due to the vagueness and expansiveness of such an idea. Additionally, our plan to add a pulse sensor as an additional input device did not come to fruition despite some successful experimenting due to a lack of time.

## 4.3 Future Works

The applications and interaction possibilities for brain computer interfaces are endless. Future areas of study may involve measuring emotional responses, and acting on these observations or evaluating if an action undertaken by the system satisfies a user.

In terms of the future of the project for Augsburg students, spellers (BCI-based applications wherein a large grid showing letters of the alphabet is displayed) and P300 potentials remain a largely unexplored area of study.

In terms of people that might actually benefit from this technology, disabled persons will be able to have more control over their surroundings, increasing their independence.

## 5 References

[1] *Telemetry Biopotential (ECG, EEG, EMG) | Millar*. URL: https://millar.com/research/applications/telemetry-biopotential.

[2] Hisaya Tanaka and Hiromi Sakata. “Measurement of the Characteristics for BCI by SSVEP”. In: *HCI International 2013 - Posters’ Extended Abstracts: International Conference, HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part I*. Ed. by Constantine Stephanidis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 623–627.

[3] *Cyton Biosensing Board (8-channels) – OpenBCI Online Store*. URL: https://shop.openbci.com/collections/frontpage/products/cyton-biosensing-board-8-channel?variant=38958638542.

[4] *Spotify menu to register applications*. URL: https://developer.spotify.com/my-applications.

[5] *Basic tutorial to use the spotify android SDK*. URL: https://developer.spotify.com/technologies/spotify-android-sdk/tutorial/.

[6] Ingmar Wellach. *Praxisbuch EEG*. 2nd ed. Stuttgart: Thiem, 2015, pp. 56–58.

[7] *ValueAnimator documentation*. URL: https://developer.android.com/reference/android/animation/ValueAnimator.html.

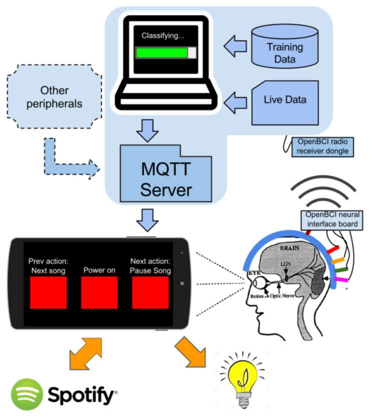
[8] *Spotify documentation*. URL: https://spotify.github.io/android-sdk/player/.

## 6 Addendum

### 6.1 List of Figures

## 6.2 Project Overview

**Projekt: Wo isn des Hirn 2.0 ?**



**Anwendungsfälle:**

- Ein- und Ausschalten einer WLAN-Steckdose
- Abspielen & Anhalten von Liedern einer Spotify- Playlist

**Ergebnisse:**

- Algorithmus für den Classifier wurde verbessert
- Elektrodenhalter wurden modifiziert und verbessert
- App zur Steuerung von Geräten aus Smart-Home wurde entwickelt

**Eingesetzte Technologien:**

- Open BCI Kit (Board & Haube) incl. Elektroden
- OpenVibe in Verbindung mit Python zur Erstellung von Classifiern
- Open BCI Software zur Visualisierung der Gehirnströme
- WLAN-Steckdosen und Spotify Developer API
- MQTT Protokoll zur Übertragung der Daten

**Projektpartner:**

- University of Ullster Technology (Leo Gallway)
- Dr. Nauman (Neurologie; Zentralklinikum Augsburg) & Fachpersonal

**Projektdaten:**

- Start: 15.03.17 bis Ende: 20.07.17
- Geschätzer Aufwand: 14 MannMonate

**Ausblick für zukünftige Aktivitäten:**

- Einsatz des entwickelten stabilen Classifiers für andere Anwendungen im Smart-Home bzw. evtl. Anwendungen für körperlich- eingeschränkte Menschen zu entwickeln

Figure 13: Overview of the entire project