Upravljanje Mislima

ANALIZA I INTERPRETACIJA EEG SIGNALA

Šegota, Žuškin | Asistivna Tehnologija | 24.1.2018.

Sadržaj

1. UVOD

Ovim seminarskim radom će biti predstavljeno prikupljanje Elektroencelografskih (EEG) signala korištenjem uređaja EMOTIV EPOC+ i softvera OpenVibe. Također će biti prikazana njihova obrada i korištenje korištenjem programskog jezika Python, paketima NumPy i SciPy.

Uređajima za prikupljanje i obradu moždanih signala moguće je analizirati moždanu aktivnost subjekta mjerenja. Ovo nam omogućava interpretaciju stanja subjekta bez potrebe za verbalnom i drugim formama ne-verbalne komunikacije sa subjektom što može biti veoma bitno kod osoba s teškoćama koje ne mogu komunicirati drugim metodama. Ovakav način komunikacije mogao bi se koristiti kod osoba sa teškoćama za prepoznavanje specifičnih uzoraka moždanih valova. Oni mogu upućivati na stanje u kojemu se osoba nalazi – poput umora ili stresa, te na temelju toga automatski raditi na otklanjanju takvih tegoba kod osoba s teškoćama ili alarmiranju drugih osoba (poput medicinskog osoblja) koje osobi s tegobama mogu pomoći.

U ovome će radu biti dan pregled EMOTIV EPOC+ uređaja, kao i osnove analize EEG signala. Finalno će biti predstavljen kod za prepoznavanje signala određenih frekvencija korištenjem EMOTIV EPOC uređaja u stvarnom vremenu.

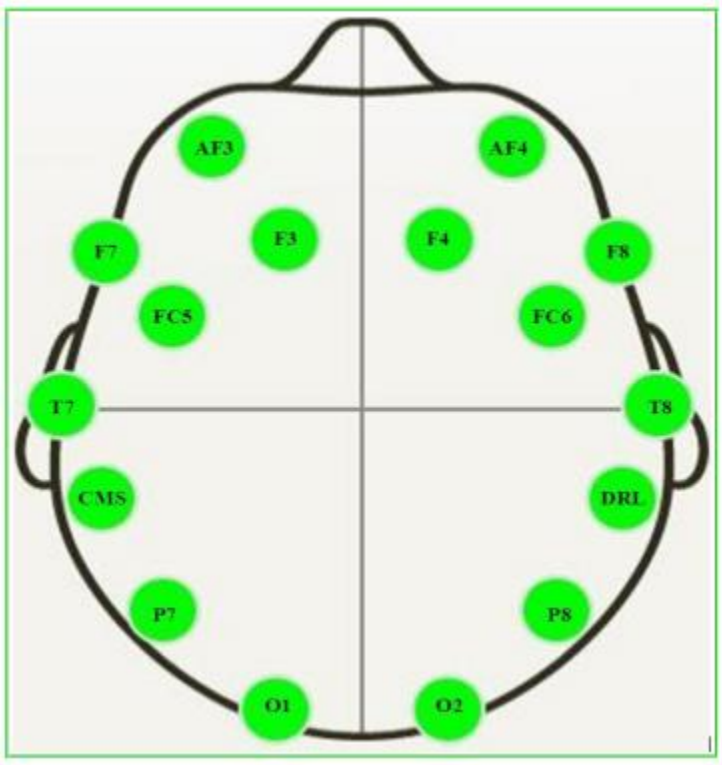
2. EMOTIV EPOC

EPOC+ je neuroheadset proizvođača Emotiv namijenjen za prikupljanje i istraživanje EEG signala te korištenje u naprednim sučeljima računalo-mozak - Brain Computer Interface (BCI) aplikacijama. EPOC+ se na računalo spaja bežično korištenjem Bluetooth Smart tehnologije ili posebnim bežičnim protokolom dizajniranim od strane Emotiva. Napaja se korištenjem interne Litijum-polimerske baterije od 640 mAh koja može omogućiti do 12 sati korištenja. Jedna od najvećih prednosti ovog uređaja je njegova bežičnost, koja omogućuje slobodu kretanja korisnika ili korištenje u situacijama gdje je pristup krajnjem korisniku žicama otežan ili nemoguć. EPOC je uređaj koji je moguće koristiti na velikom broju sustava:

- Windows XP, Vista, 7, 8, 10
- Linux (Ubuntu, Fedora)
- Mac OS X,
- Android 4+

Slika 1 Emotiv EPOC EEG neuroheadset

Emotiv omogućuje prikupljanje signala sa 14 lokacija na lubanji korištenjem 14 senzora, uz 2 referentne točke. Uređaj koristi sekvencijsko prikupljanje frekvencijom od 128 bita sa jednim Analogno digitalnim konverterom (ADC). Uređaj je u mogućnosti prikupljati signale između 0.2 i 45 Hz sa ugrađenim filterima na 50 i 60 Hz. Lokacije prikupljanja signala prikazane su na slici.



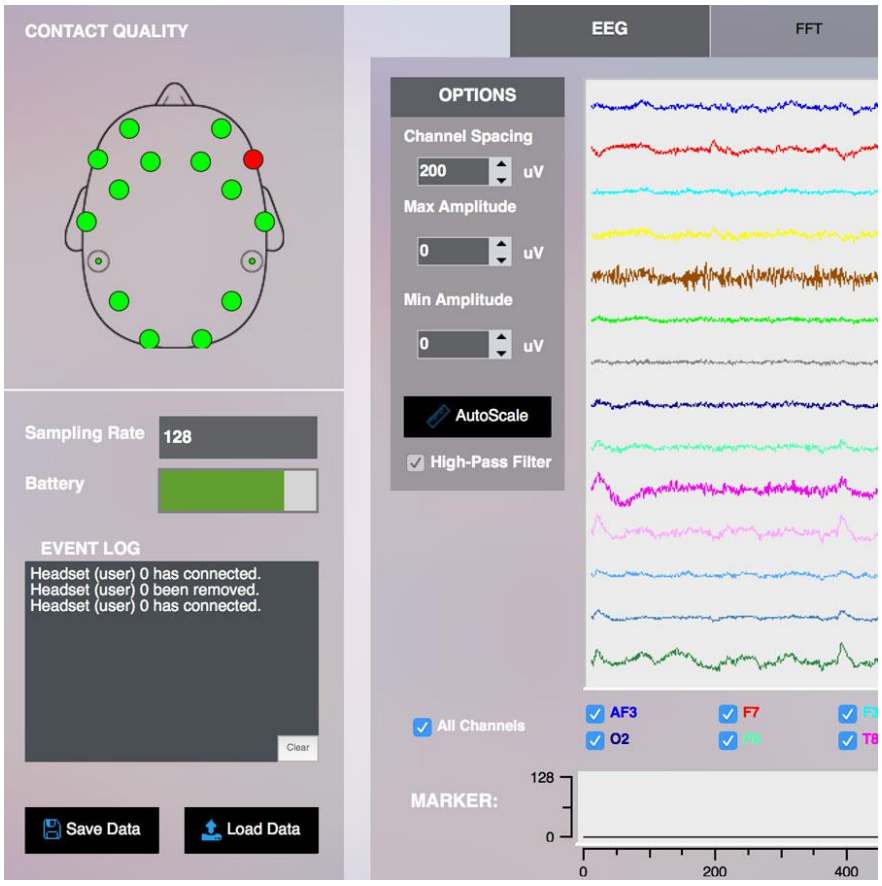
Slika 2 Lokacije prikupljanja EEG signala

2.1. SPAJANJE SA RAČUNALOM I KORIŠTENJE ZA PRIKUPLJANJE EEG SIGNALA

Kako bismo koristili EPOC potrebno ga je povezati sa računalom. Njegovo je povezivanje izvršeno putem bežičnog USB dongle uređaja koji se povezuje sa Emotiv EPOC uređajem. Instalacija potrebnih drivera za USB dongle je automatska.

Na uređaj postavimo senzorske elektrode na predviđena mjesta (pipke uređaja) te na njih postavimo spužvice koje smo prethodno namočili fizikalnom

otopinom elektrolita kako bismo omogućili provođenje signala do elektrode. Kako bismo utvrdili je li povezivanje bilo uspješno potrebno je pokrenuti program za vizualizaciju signala – Emotiv Xavier EEG. On omogućuje prikaz očitanih signala i pokazuje koje su od elektroda ispravno postavljene i primaju kvalitetan signal. Loše postavljene elektrode je potrebno dodatno podesiti – ili dodatnim namakanjem fizikalnom otopinom ili podešavanjem pozicije na glavi korisnika. Ako elektroda ne prima signal, njena će lokacija na slici biti označena crnom bojom. U slučaju slabog ili lošeg signala, lokacija će na slici biti označena narančastom ili crvenom bojom. Zelenom se bojom označavaju elektrode koje su dobro postavljene. Ovo je vidljivo na donjoj slici gdje su sve elektrode osim elektrode F8 dobro postavljene.



Slika 3 Prikaz Emotiv Xavier EEG programa

Kada smo uređaj uspješno postavili i povezali sa računalom možemo krenuti u obradu i analizu signala. Moguće je i snimiti signale te ih kasnije obrađivati ili raditi sa njima u stvarnom vremenu.

3. Korišteni softver

U ovome poglavlju biti će opisan softver koji je korišten za obradu i analizu EEG signala dobivenog korištenjem Emotiv EPOC uređaja opisanog u prethodnom poglavlju.

Na početku izrade seminarskog rada je za obradu prethodno snimljenih signala korišten MATLAB u kojemu je rađen spektrogramski prikaz kanala snimljenih signala.

Kasnije je se, za potrebe analize i obrade u stvarnom vremenu, koristio program OpenVibe koji omogućuje akviziciju signala. Također je korišten programski jezik Python za obradu primljenih signala. Detalji korištenih funkcija i programski kodovi opisani su u sljedećem poglavlju. Dalje slijedi kratki opis korištenih programa, programskih paketa i načina rada sa njima.

3.1. MATLAB

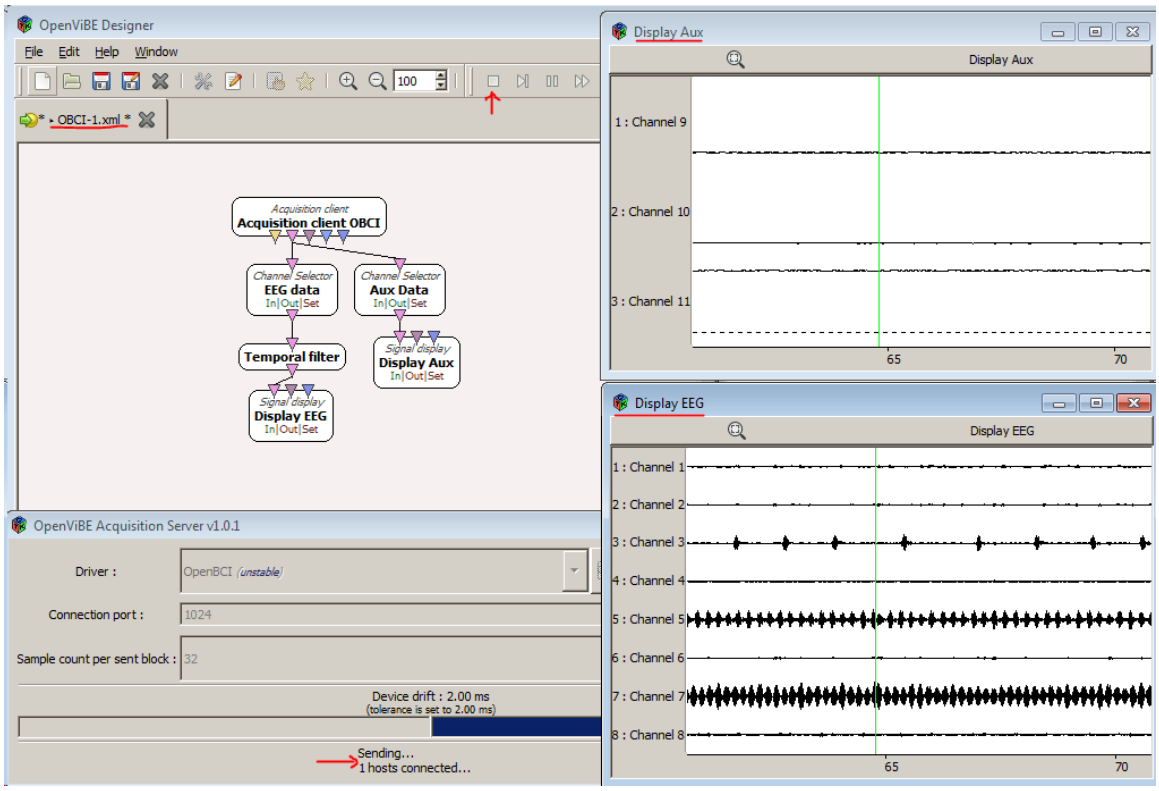
MATLAB je numeričko programsko okruženje optimizirano za rad sa matricama. Uz njega je korišten i modul Signal Processing Toolbox koji omogućava obradu, filtriranje, analizu, generiranje i prikaz kompleksnih signala.

Iako OpenVibe ima module koji predviđaju spajanje sa MATLAB skriptama, tokom izrade ovog rada nismo uspjeli uspješno povezati ova dva programska okruženja, radi čega je korišten Python.

3.2. OpenVibe

OpenVibe programski je paket otvorenog koda za kreiranje sučelja mozak-računalo koji podržava rad na Windows i Linux okruženjima. OpenVibe omogućava akviziciju, obradu i vizualizaciju EEG signala u stvarnom vremenu. OpenVibe koristi grafičku paradigmu programiranja i omogućava kreaciju sučelja korištenjem velikog broja postojećih blokova, kao i razvoj vlastitih blokova korištenjem programskih jezika C++, Python i MATLAB.

OpenVibe se sastoji od nekoliko programa. U svrhu izrade ovog seminarskog rada korišteni su: OpenVibe Designer i OpenVibe Acquisition server. Acquisiton Server omogućuje povezivanje sa Software Development Kit (SDK) programskim paketima raznih uređaja za akviziciju signala, a samim time i tim uređajima. Acquisition Server omogućuje korištenje primljenih podataka sa uređaja u OpenVibe okruženju prosljeđujući ih kroz modul Acquisition Client. U njemu je također omogućena osnovna konfiguracija postavki uređaja. OpenVibe Designer se koristi za izgradnju sučelja, te omogućuje prikaz signala i poruka koje sučelje obrađuje i/ili generira. Za potrebe ovog rada korištena je OpenVibe verzija 1.2.2. zbog toga što najnovija verzija (2.0.0) nije u mogućnosti povezati se sa uređajem Emotiv EPOC. Najnoviju verziju je moguće povezati samo sa novijim Emotiv EPOC+. Izgled OpenVibe programa je vidljiv na slici:



Slika 4 Sučelje OpenVibe Acquisition Server i Designer sa blokovima za prikaz signala

3.2.1. Akvizicija signala Emotiv EPOC uređaja

Kako bismo povezali OpenVibe i Emotiv EPOC uređaj potrebno je instalirati potreban SDK – Emotiv Research SDK verzija 2.0.0. Nakon instalacije potrebno je pokrenuti OpenVibe Acquisition Server i odabrati driver koji odgovara uređaju koji koristimo te postaviti port za konekciju (u našem slučaju na 1024). Zatim je potrebno otvoriti meni Driver Properties te u njemu postaviti lokaciju instaliranog SDK (uobičajeno C:\Program Files(x86)\Emotiv Research SDK\dll\32 bit) – direktorij koji sadrži edk.dll. Kada je to učinjeno potrebno je kliknuti na gumb Connect. Ukoliko nije moguće uspostaviti konekciju, Acquisition Server će ispisati pogrešku u konzoli. Ako je veza uspostavljena Acquisition server će na dnu prikazati tekst „1 host connected“.

Kako bismo podatke koristili, u OpenVibe Designeru je potrebno dodati modul Acquisition Client u kojemu je dovoljno postaviti ranije odabrani port(1024) te ga povezati sa daljnjim blokovima za obradu ili prikaz signala.

3.3. PYTHON

Python je interpreterski programski jezik visoke razine opće namjene. Uz sam Python su, za potrebe izrade ovog rada, korištene i knjižnice funkcija NumPy i SciPy.

NumPy je osnovni paket funkcija za znanstveno računanje koji omogućava korištenje N-dimenzionalnog polja sa metodama za računanje i obradu vrijednosti spremljenih u njemu kao i mnoge funkcije linearne algebre i slučajnih brojeva.

SciPy je okruženje otvorenog koda koje omogućava korištenje Pythona kao alata za obradu matematičkih, inženjerskih i znanstvenih problema. Nama su posebno u interesu funkcije za obradu i filtriranje signala.

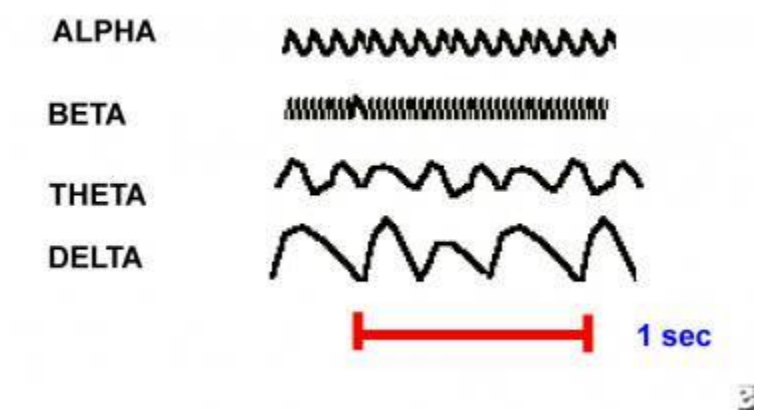
3.3.1. Korištenje Pythona u sklopu OpenVibea

Kako bi se omogućio rad sa OpenVibe programskim paketom, potrebno je koristiti verziju Pythona 2.7. Također je potrebno koristiti 32 bitni Python i potrebne knjižnice – NumPy i SciPy. Također se preporuča instalacija Pythona u zadani direktorij – C.\Python27 radi jednostavnosti. Ako je Python podešen u drugom direktoriju moguće je da će biti potrebno dodatno postavljanje PATH varijabli Windows okruženja kako bi OpenVibe mogao pokretati Python skripte.

Nakon instalacije potrebno je u OpenVibe Designeru koristiti Python Scripting modul. U Python Scripting modulu zadajemo Python datoteku koja će se koristi tokom pokretanja te podešavamo broj ulaza i izlaza u modul iz menija koji otvaramo desnim klikom na isti. Ako je sve uspješno postavljeno, OpenVibe će u blok prosljeđivati podatke i očitavati ih iz izlaza bloka ako je potrebno te će se skripta koju smo zadali u bloku izvršavati.

4. Obrada EEG signala

Obrada se temelji na analiziranju signala i traženju određenih elemenata u signalu. Ovi se signali kategoriziraju u nekoliko vrsta. Najzanimljiviji su nam prikazani na slici:



Slika 5 Prikaz izgleda nekih od čestih moždanih valova

Frekvencije su ovih valova prikazane u tablici.

Vrsta Vala	Frekvencija [Hz]
Alpha	0.1 – 4
Beta	4-8
Theta	8-13
Delta	13-30

Tablica 1 Frekvencije čestih vrsta moždanih valova

Alfa se valovi najčešće detektiraju kod opuštenih odraslih subjekta. Beta se valovi pojavljuju kod stimulacija osjetila, posebice kod svjesne aktivnosti poput

govora, rješavanja problema i donošenja odluka. Theta valovi su povezani sa podsvjesnom aktivnosti. Ove se vrsta valova najčešće javljaju kada je subjekt opušten. Delta se valovi najčešće detektiraju u dubokom snu te su abnormalni kod budnih odraslih subjekta.

Radi jednostavnosti, u ovome se radu pretraživanje značajki signala vrši po frekvenciji signala, gdje se određene frekvencije filtriraju band-pass filterom i mjeri se njihov intenzitet. Na temelju toga zaključuje se postojanje određene vrste aktivnosti. Kako bi se detaljnije precizirale točne značajke prikupljene metodama opisanima u radu, bio bi potreban veliki broj mjerenja, što će biti moguće ako se nastavi rad na ovome projektu u duljem vremenskom razdoblju nego pruženom.

5. Obrada EEG signala u MATLABu

Tokom izrade rada izvršena su snimanja EEG signala moždane aktivnosti članova grupa koristeći Emotiv EPOC uređaj i Emotiv Xavier EEG program. Tokom snimanja subjektima su postavljana raznolika pitanja kako bi se potakla njihova moždana aktivnost. Ova snimanja su konvertirana i obrađena u MATLABu. Naša je grupa za zadatak imala izraditi spektrograme dobivenih podataka. U ovome poglavlju opisane su korištene MATLAB funkcije iz Signal Processing Toolboxa korištene za izradu spektrograma.

Kako bismo obrađivali dobivene podatke, konvertiramo ih u matrični zapis koji MATLAB može obrađivati, i učitamo u MATLAB Workspace povlačenjem mišem.

5.1. MATLAB KOD

Na početku je potrebno postaviti vrijednosti koje ćemo koristiti tokom izrade spektrograma. Vrijednosti su preuzete prema naputku iz reference [7]. Stoga je frekvencija uzorkovanja fs postavljena na 128 Hz pošto je to frekvencija uzorkovanja Emotiv EPOC uređaja, trajanje jednog segmenta za Fourierovu transformaciju korištenjem Fast Fourier Transform (FFT) algoritma postavljeno je na 64 uzorka, a preklapanje segmenata za izradu spektrograma je postavljeno na polovicu trajanja uzorka za FFT.

```
fs=128
segment_length=64
sample_overlap=segment_length/2
```

Zatim je potrebno postaviti vrijednosti za iscrtavanje grafa spektrograma, poput naslova, veličine teksta i slično:

```
f = figure
p = uipanel('Parent',f,'BorderType','none');

p.Title = 'Spektrogram EEG vrijednosti - subjekt: Sandi';
p.TitlePosition = 'centertop';
p.FontSize = 12;
p.FontWeight = 'bold';
```

Potom ulazimo u for petlju koja se kreće po vrijednostima od 1 do 14 sa korakom od 1. Ovo je napravljeno kako bismo obradili svaki od 14 kanala zasebno. Na početku svake nove petlje pomičemo položaj novog grafa spektrograma na ukupnom grafu na sljedeću poziciju, izračunavamo i crtamo spektrogram te postavljamo redni broj kanala kao naslov.

```
for i=1:1:14
    subplot(4,4,i, 'Parent', p)
    spectrogram(SandiEEGadapt.data(i, 1:30816), segment_length,
sample_overlap, segment_length, fs, 'yaxis')%using hamming window
    title(['Kanal ' num2str(i) '.'])
    %view(-77,72)
end
```

Bitno je obratiti pozornost na funkciju za izračunavanje i iscrtavanje spektrograma:

```
spectrogram(SandiEEGadapt.data(i, 1:30816), segment_length,
sample_overlap, segment_length, fs, 'yaxis')
```

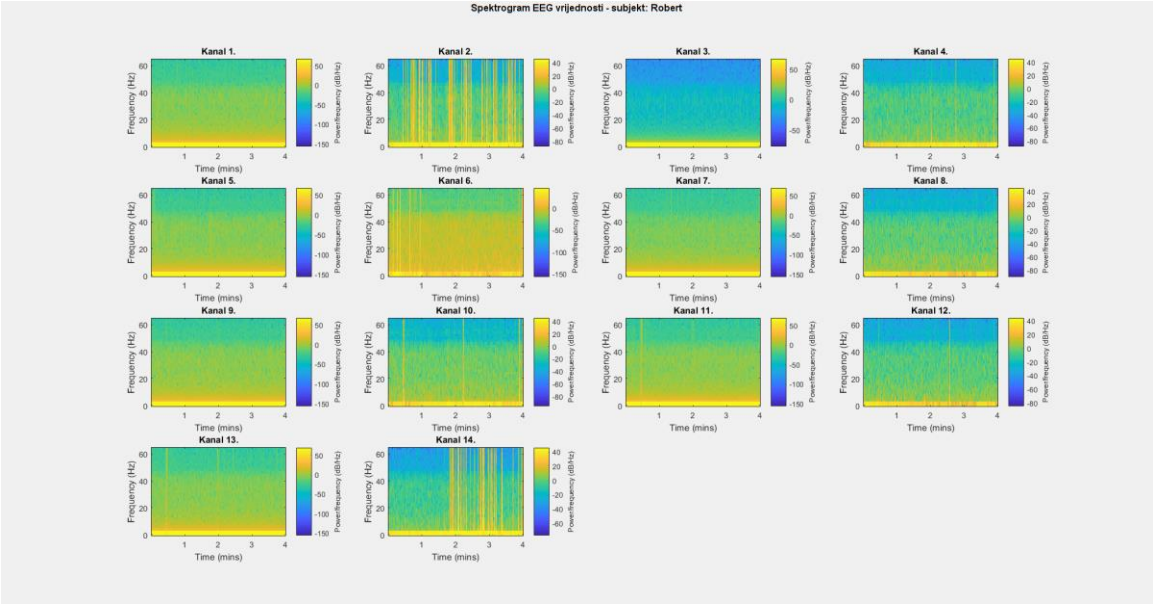
Vidljivo je da ona uzima argumente *fs*, *segment_length* i *sample_overlap* koje smo ranije definirali, ali uzima u obzir i druge argumente. Tako argument *'yaxis'* definira uz koju će os spektrogram biti crtan, a prvi argument *segment_length* definira koji prozor koristimo za obradu signala. Ovdje prema [7] koristimo Hammingov prozor. Pod taj je argument potrebno zadati segment_length, što je kod funkcije spectrogram ekvivalentno korištenju funkcije *hamming(segment_length)*. Ako bismo željeli koristiti neki drugi prozor, primjerice popout trokutastog, tada bi umjesto prvog *segment_length* kao argument postavili funkciju *triang(segment_length)*. Prvi argument u spectrogram funkciji je *SandiEEGadapt.data()*. Ovo su podaci mjerenja koje smo učitali u MATLAB u matričnom obliku. Tako argumenti metode data tog objekta definiraju koji redak i stupce koristimo. Ovdje je redak zadan kao varijabla. Kako jedan redak u matrici podataka predstavlja jedan kanal, u svakoj se iteraciji for petlje analizira po jedan kanal podataka. Drugi argument čine elementi vektora, odnosno stupci matrice

podataka iz retka koji smo zadali kao prvi argument. U ovome slučaju uzimamo prvih 30816 elemenata za analizu, odnosno: $t = \frac{n}{f_s} = \frac{30816}{128Hz} = 240.75s = 4.0125min$ snimanja.

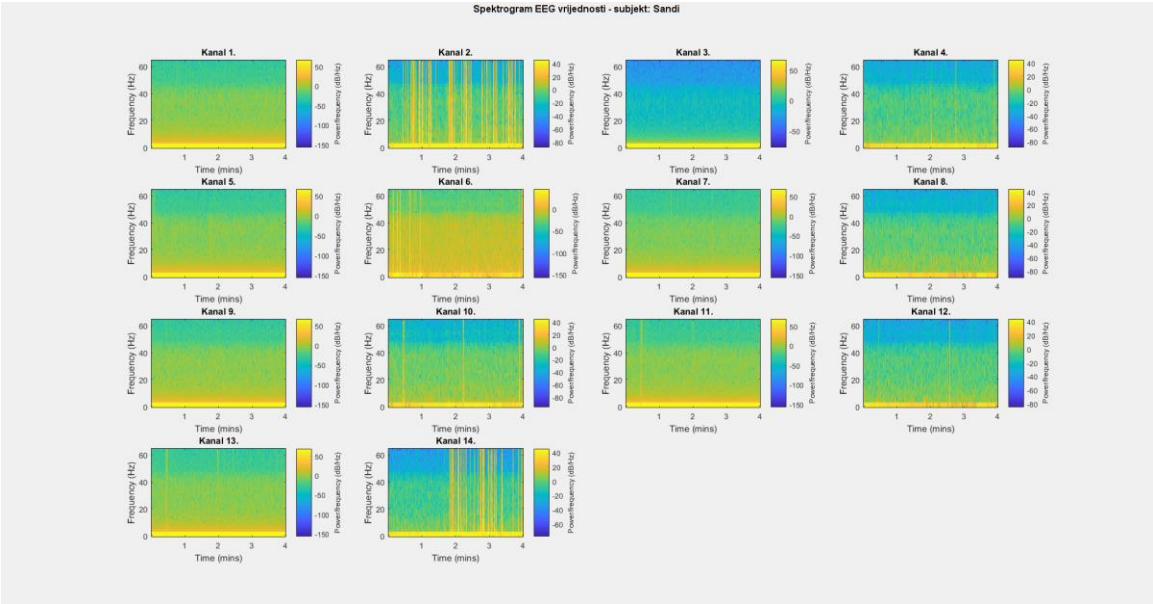
Kompletan kod nalazi se u Prilogu 1.

5.2. SPEKTROGRAMI

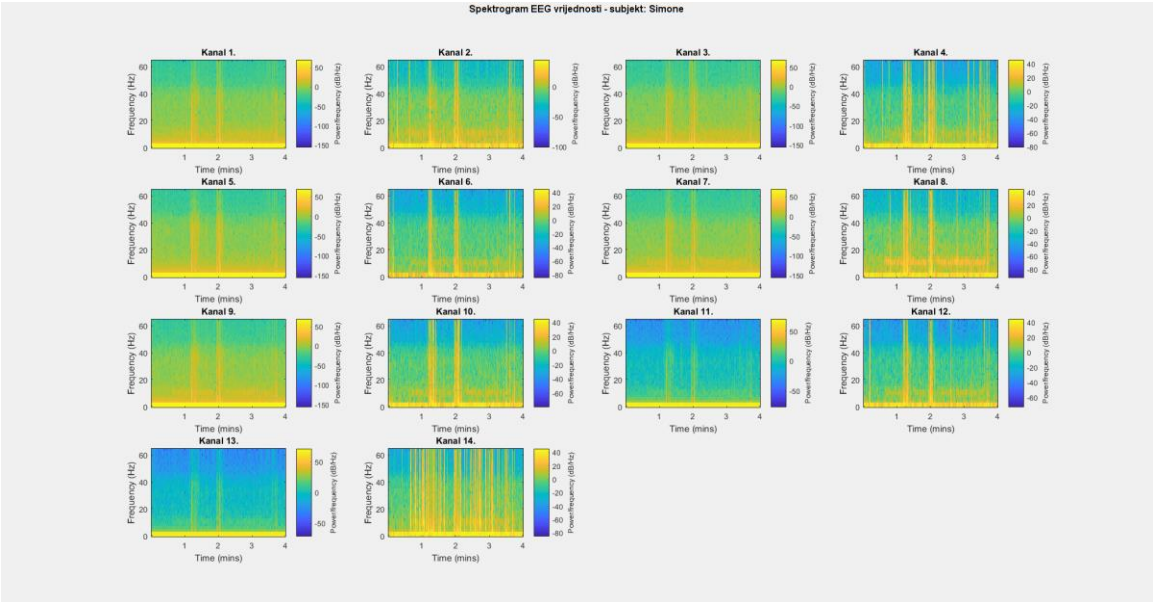
Spektrogrami kreirani korištenjem prethodno opisanog koda prikazani su na slikama 6, 7, i 8. Na slikama su vidljivi porasti aktivnosti u EEG signalima subjekata kada su im davani poticaji u obliku verbalno postavljenih pitanja i mogućih odgovora (npr. „U kojem gradu živiš?“ „Pula“… „Rijeka“… „Split“).



Slika 6 Spektrogram EEG signala - subjekt Robert (M, 23)



Slika 7 Spektrogram EEG signala - subjekt Sandi (M, 22)



Slika 8 Spektrogram EEG signala - subjekt Simone (M, 22)

6. Izrada sučelja mozak-računalo

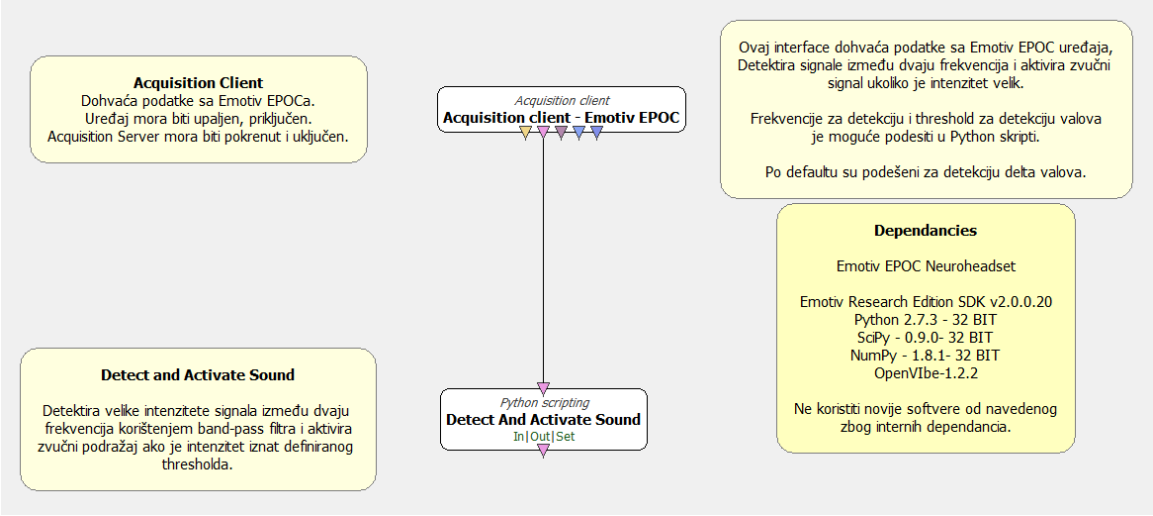
Kao cilj rada planirana je izrada sučelja koje će korištenjem Emotiv EPOC uređaja i Pythona detektirati povećanje intenziteta u određenom frekvencijskom pojasu i aktivirati neku vrstu poticaja. Opis izrađenog sučelja u nastavku je poglavlja.

6.1. OPENVIBE SUČELJE

Izrađeno OpenVibe sučelje je jednostavno, i sastoji se od dva bloka. Prvi blok je Acquisition Client koji dohvaća podatke sa Acquisition Servera – odnosno Emotiv EPOC uređaja, te ih prosljeđuje u daljnje blokove.

Sljedeći je blok Python scripting blok koji nam omogućava pokretanje Python skripte koju smo napisali. Python skriptu koju ćemo pokretati definiramo dvostrukim klikom na blok i unosom putanje do nje.

Uz ove blokove moguće je i dodati blokove za prikaz signala, ali smo odlučili ne dodavati ih radi izbjegavanja opterećenja i usporavanja sustava – želimo da naš sustav radi u pravom vremenu.



Slika 9 Prikaz izrađenog sučelja u OpenVibeu

6.2. PYTHON PROGRAM

Obradu signala dobivenog sa Emotiv EPOCa rađena je u Pythonu. Iako je u početku planirana izrada ove skripte u MATLABu, radi neuspjeha u povezivanju MATLABa i OpenVibea odlučeno je korištenje Pythona. Odlučili smo ga koristiti zbog više razloga – jednostavno povezivanje sa OpenVibe sučeljem, a uz to je Python jednostavan programski jezik sa alatima za obradu signala korištenjem knjižnica funkcija NumPy i SciPy. Faktor je bio i dobro ranije poznavanje programskog jezika.

Izrađena skripta sastoji se od 2 dijela. Prvi je dio za dohvaćanje akviziranog signala sa uređaja na ulazu u OpenVibe blok, a drugi je dio za filtriranje i obradu signala i aktivaciju zvučnog signala.

6.2.1. Očitavanje ulaza

Prvobitno je bilo potrebno uvesti funkcije iz knjižica koje ćemo koristiti. Ovo radimo naredbom *import*. Uvodimo ranije spomenute knjižice numpy i scipy, s time da iz scipy uvodimo samo određene funkcije iz podpaketa signal koje ćemo kasnije koristiti. Ovo radimo jer je scipy velika knjižica i uvođenje svih funkcija bi povećalo memorijsko opterećenje programa. Uz njih uvodimo i knjižicu winsound koja nam omogućava generiranje zvučnog signala.

```
import numpy
import winsound
from scipy.signal import butter, lfilter, freqz
```

Nakon uvođenja knjižica kreiramo klasu MyOVBox. Ovo je klasa koja se tokom izvođenja instancira u objekt sa kojim OpenVibe sučelje može komunicirati. U njoj je potrebno kreirati dvije funkcije: _init i process.

```
class MyOVBox(OVBox):

    def __init__(self):
        OVBox.__init__(self)
        self.signalHeader = None

    def process(self):
```

Kao što je vidljivo iz koda ovih funkcija one izvršavaju operacije nad klasom – stoga koristimo argument self. Metoda _init inicira pokretanje i stvaranje objekta OVBox te postavlja sinalHeader samog objekta na prazan. Varijabla signalHeader sadržava podatke o primljenom signalu koji se postavljaju u funkciji process(self), stoga ga trenutno inicijaliziramo na praznog kako se prilikom pokretanja ne bi učitali podaci iz prošlog pokretanja ili slučajne memorijske lokacije.

Samu obradu signala vršimo u funkciji process(self). Na početku funkcije započinjemo for petlju koja nam omogućava obradu cijelog primljenog dijela signala. Iteriramo po svakom chunkIndexu – chunkIndex je jedan uzorak signala, za cijelu duljinu primljenog signala – koju dobijemo korištenjem funkcije len(self.input[0]), gdje je self.input[0] polje koje sadrži sve ulazne signale. Input je ugrađeni element OpenVibe sučelja i on je vektor matrica koji sadrži matrične zapise ulaznih signala – tako je self.input[0] u našem slučaju zapravo matrica sa 14 redaka koja sadrži naše kanale.

```
for chunkIndex in range( len(self.input[0]) ):
```

Zatim je potrebno postaviti signalHeader koji smo ranije spomenuli. U signalHeader su sadržani svi podaci o signalu, koji je prvi element signala te sučelje šalje u blok. Provjeru za utvrđivanjem njegovog slanja detektiramo provjerom da li je element input vektora tipa OVSignalHeader

```
if(type(self.input[0][chunkIndex]) == OVSignalHeader):
```

i ako jest postavljamo signalHeader trenutne instance objekta MyOVBox kao te vrijednosti.

```
self.signalHeader = self.input[0].pop()
```

Naredbom pop() mičemo signalHeader podatke iz signala kako bismo imali čisti signal za obradu. Zatim vrijednosti signala poput vremena početka i kraja te frekvencije uzorkovanja postavljamo u varijablu outputHeader te naredbom append() te vrijednosti dodajemo na početak izlaznog vektora output.

```
outputHeader = OVSignalHeader(
```

```
self.signalHeader.startTime,
self.signalHeader.endTime,
[1, self.signalHeader.dimensionSizes[1]],
['Mean']+self.signalHeader.dimensionSizes[1]*[''],
self.signalHeader.samplingRate)

self.output[0].append(outputHeader)
```

Može se primijetiti da u našem sučelju ne koristimo nikakve izlaze, što je vidljivo na slici 9, pri čemu se postavlja pitanje zašto je ovdje programiran izlaz? Razlog je tome što, iako nam taj izlaz trenutno nije potreban, odlučili smo ga izraditi i ostaviti kako bismo omogućili daljnje proširenje mogućnosti sučelja. Također nam omogućuje i jednostavno provjeravanje grešaka dodavanjem blokova za vizualizaciju u OpenVibe sučelje.

Dalje u kodu provjeravamo da li su podaci koje dobivamo signal. Koristeći naredbu koja uspoređuje ulazne podatke sa tipom podataka signala OpenVibea vrši se provjera podataka.

```
elif(type(self.input[0][chunkIndex]) == OVSignalBuffer):
```

Ako je, tada te podatke ponovno naredbom pop() postavljamo u varijablu chunk koju potom pohranjujemo u numpy.array polje. NumPy array je tip podataka polja koje omogućuje rad sa mnogim numpy funkcijama nad podacima spremljenima u njega. Tako je sada moguće reformirati te podatke prema dimenzijama signala koje su spremljene u header. Kada je to učinjeno ponovno podatke prebacujemo u chunk te ih dodajemo na izlazni signal.

```
chunk = self.input[0].pop()
numpyBuffer =
numpy.array(chunk).reshape(tuple(self.signalHeader.dimensionSizes))
chunk = OVSignalBuffer(chunk.startTime, chunk.endTime,
numpyBuffer.tolist())
self.output[0].append(chunk)
```

Na kraju detektiramo da li je primljeni dio simbol za kraj signala, i ako jest, dodajemo ga na kraj našeg izlaznog signala.

```
elif(type(self.input[0][chunkIndex]) == OVSignalEnd):
                self.output[0].append(self.input[0].pop())
```

Time je završen dio koda za očitavanje signala iz sučelja, kojega je potrebno dodatno obraditi.

6.2.2. Obrada signala u Pythonu

Nakon koda opisanog u prošlom potpoglavlju, unutar ranije započete for petlje pišemo kod za obradu podataka. Ovaj kod se može podijeliti u 2 dijela – filtriranje i generiranje zvučnog koda.

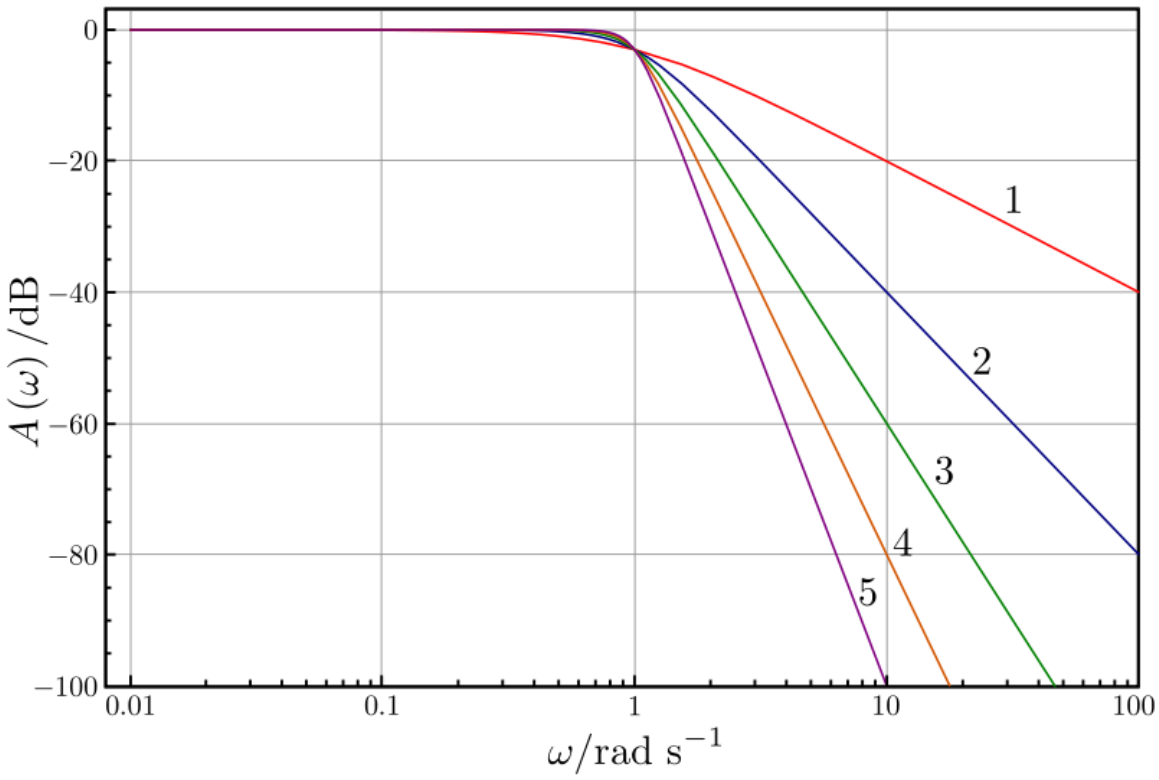
Na početku je potrebno postaviti vrijednosti varijabli koje ćemo koristiti u filtriranju:

```
fs=128
order=6
nyq = 0.5 * fs
cutoff=30.0
normal_cutoff = cutoff / nyq
cutoff2=13.0
normal_cutoff2 = cutoff2 / nyq
```

Ove vrijednosti su redom frekvencija uzorkovanja signala, red filtra koji ćemo koristiti i Nyquistova frekvencija. Zatim postavljamo frekvencije odrezivanja u varijable cutoff (za gornju granicu frekvencije) i cutoff2 (za donju granicu frekvencije), te ih normaliziramo tako što ih podijelimo Nyquistovom frekvencijom.

6.2.2.1. Butterworthov filtar

Za filtriranje signala korišten je Butterworthov filter. Butterworthov filter, također poznat kao maksimalno ravni magnitudni filter, je filter koji je dizajniran kako bi imao što ravniji frekventni odziv u području signala koji propušta. Frekventni odziv takvog filtra, za različite redove je vidljiv na slici 10.



Slika 10 Frekventni odziv Butterworthovog filtra različitih redova

Dobitak Butterworthovog filtra je dan formulom:

$$G^2(\omega) = |H(j\omega)|^2 = \frac{G_0^2}{1 + \left(\frac{j_\omega}{j_{\omega_c}}\right)^{2n}}$$

Gdje su:

n red filtra

ω_c frekvencija odrezivanja

G_0 istosmjerni dobitak

Ako želimo odrediti prijenosnu funkciju $H(s)$, gdje je $s = \sigma + j\omega$, korištenjem pravila Laplaceove transformacije $|H(s)|^2 = H(s)H(\acute{s})$i $H(-j\omega) = H(\acute{j}\omega)$ dobijemo da vrijedi:

$$H(s)H(-s) = \frac{G_0^2}{1 + \left(\frac{-s^2}{j\omega_c}\right)^n}$$

Pošto za svaki k-ti pol vrijedi

$$\frac{-s_k^2}{\omega_c^2} = (-1)^{\frac{1}{n}} = e^{\frac{j(2k-1)\pi}{n}} \underset{\rightarrow}{yields}\ s_k = \omega_c e^{\frac{j(2k+n-1)\pi}{2n}} \quad , za\ k = 1,2,3, \dots, n$$

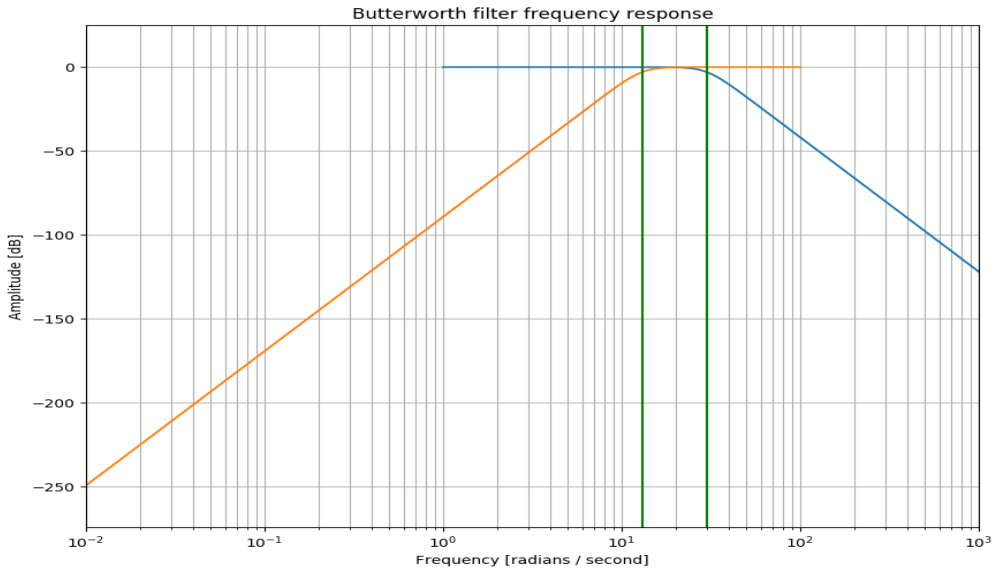
Dobijemo da prijelazna funkcija može biti zapisana kao

$$H(s) = \frac{G_0}{\prod_{k=1}^{n} \frac{s - s_k}{\omega_c}}$$

Butterwhortov filtar je u scipy knjižnici implementiran u funkciji butter(). Kako bismo je koristili pozovemo je i zadamo joj red filtra order, normaliziranu frekvenciju odrezivanja i tip filtra (odnosno hoće li propuštati niske ili visoke frekvencije). Funkcija butter() nam vraća dvije vrijednosti b i a koje koristimo za filtriranje signala spremljenog u numpyBuffer funkcijom lfilter(), prvo nad numpyBufferom, a zatim nad već filtriranim y.

```
b, a = butter(order, normal_cutoff, btype='low', analog=False)
y = lfilter(b, a, numpyBuffer)
b, a = butter(order, normal_cutoff, btype='high', analog=False)
y = lfilter(b, a, y)
```

Kako bi dobili band-pass filter, signal prvo filtriramo low-pass filterom koji propušta sve frekvencije ispod određene. Zatim signal provlačimo kroz high pass filtar koji eliminira sve frekvencije niže od određene. Obje vrijednosti spremamo u y. Frekventni odziv ovakvog filtra vidljiv je na slici gdje se propuštaju frekvencije između dvaju zelenih linija. High pass filtar je prikazan narančastom, a low pass zelenom bojom:



Slika 11 Frekventni odziv korištenog Butterworth filtra

Na samom kraju vrijednosti signala pretvorimo u njihove absolutne kako se ne bi međusobno poništavale, zbrojimo ih te taj intenzitet normaliziramo.

```
freq=int(numpy.fabs(int(numpy.floor(y.sum()))-138000))
```

Ako je ta vrijednost veća od zadane (koja je utvrđena eksperimentalno) aktiviramo zvučni signal. Prvi argument kod zvučnog signala je frekvencija zvuka, a drugi je trajanje signala u milisekundama

```
if(freq>2000):
      winsound.Beep(440, 300)
```

Zatim je još samo potrebno instancirati objekt kreirane klase:

```
box = MyOVBox()
```

7. Zaključak

Kodom i sučeljem izrađenim u sklopu ovog rada možemo detektirati poraste intenziteta signala u određenom rasponu frekvencija i na njih reagirati korištenjem zvučnog signala. Kako bismo poboljšali točnost detekcije bilo bi potrebno izvesti veliki broj mjerenja na više različitih subjekata kako bi se utvrdili odgovarajući rasponi frekvencija za detekciju različitih stanja subjekata.

Također, bilo bi moguće izvesti naprednije sustave reagiranja od samog zvučnog sustava upozorenja. Python ima veliki broj knjižnica funkcija koje bi omogućile spajanje sa uređajima poput mikrokontrolera spojenih na druge sustave koji mogu direktno utjecati na stanje krajnjeg korisnika. Postojala bi i mogućnost izrade sustava za umrežavanje za obavještavanje i nadzor krajnjih korisnika od strane medicinskog osoblja.

Sustav koji je ovdje dan je primjer prvog koraka u izgradnji sustava za pomoć osobama sa teškoćama temeljenog na sučeljima mozak-računalo.

Reference

[1] „Emotiv EPOC“, Tehnički fakultet Rijeka; s interneta, 22.1.2018.: http://www.riteh.uniri.hr/ustroj/zavodi/zae/laboratoriji/laboratorij-za-asistivnu-tehnologiju/asistivna-tehnologija/seminari/brain-computer-interfaces/emotiv-epoc/

[2] „Emotiv EPOC Product Page“, Emotiv; s interneta, 22.1.2018.: https://www.emotiv.com/product/emotiv-epoc-14-channel-mobile-eeg/

[3] „Using MATLAB With OpenVIbe“, OpenVibe; s interneta, 22.1.2018.: http://openvibe.inria.fr/tutorial-using-matlab-with-openvibe/

[4] „How to connect Emotiv EPOC with OpenVibe“, OpenVibe; s interneta, 22.1.2018.: http://openvibe.inria.fr/how-to-connect-emotiv-epoc-with-openvibe/

[5] „Using Python with OpenVibe“, OpenVIbe; s interneta, 22.1.2018.: http://openvibe.inria.fr/tutorial-using-python-with-openvibe/

[6] „Signl Processing Toolbox“, MathWorks; s interneta, 22.1.2018.: https://www.mathworks.com/products/signal.html

[7] „EEG Recording and Analysis for Sleep Research“, Ian G. Campbell; Curr Protoc Neurosci

[8] „Analysis of Electroencephalographa (EEG) Signals and Its Categorization – A Study“, J Satheesh Kumar, P Bhuvaneswari; Sciverse ScienceDirect

[9] „SciPy Reference“, SciPy; s interneta, 22.1.2018.https://www.scipy.org/

[10] „NumPy Reference“, NumPy,; s interneta, 22.1.2018.: http://www.numpy.org/

[11] „Butterworth Filter“, Wikipedia; s interneta, 22.1.2018.: https://en.wikipedia.org/wiki/Butterworth_filter

Prilog 1

MATLAB KOD ZA GENERIRANJE SPEKTROGRAMA

```
fs=128
%Following values based on
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2824445/
%using default hamming window
segment_length=64
sample_overlap=segment_length/2

f = figure
p = uipanel('Parent',f,'BorderType','none');

p.Title = 'Spektrogram EEG vrijednosti - subjekt: Sandi';
p.TitlePosition = 'centertop';
p.FontSize = 12;
p.FontWeight = 'bold';

for i=1:1:14
    subplot(4,4,i, 'Parent', p)
    spectrogram(SandiEEGadapt.data(i, 1:30816), segment_length,
sample_overlap, segment_length, fs, 'yaxis')%using hamming window
    title(['Kanal ' num2str(i) '.'])
    %view(-77,72)
end
```

Prilog 2.

PYTHON KOD ZA DETEKCIJU INTENZITETA VALOVA I AKTIVACIJU PODRAŽAJA

```
import numpy
import winsound
from scipy.signal import butter, lfilter, freqz

class MyOVBox(OVBox):
    def __init__(self):
        OVBox.__init__(self)
        self.signalHeader = None

    def process(self):

        for chunkIndex in range( len(self.input[0]) ):
            if(type(self.input[0][chunkIndex]) == OVSignalHeader):
                self.signalHeader = self.input[0].pop()

                outputHeader = OVSignalHeader(
                self.signalHeader.startTime,
                self.signalHeader.endTime,
                [1, self.signalHeader.dimensionSizes[1]],
                ['Mean']+self.signalHeader.dimensionSizes[1]*[''],
                self.signalHeader.samplingRate)

                self.output[0].append(outputHeader)

            elif(type(self.input[0][chunkIndex]) == OVSignalBuffer):
                chunk = self.input[0].pop()
                numpyBuffer =
numpy.array(chunk).reshape(tuple(self.signalHeader.dimensionSizes))
                numpyBuffer = numpyBuffer.mean(axis=0)
                chunk = OVSignalBuffer(chunk.startTime, chunk.endTime,
numpyBuffer.tolist())
                self.output[0].append(chunk)

            elif(type(self.input[0][chunkIndex]) == OVSignalEnd):
                self.output[0].append(self.input[0].pop())

            ##Low-Pass Filter implementation
            fs=30
            cutoff=3.667
            order=6
            nyq = 0.5 * fs
            normal_cutoff = cutoff / nyq
            b, a = butter(order, normal_cutoff, btype='low',
analog=False)
            y = lfilter(b, a, numpyBuffer)

            #High Pass Filter Implementation
            cutoff2=2.3
            normal_cutoff = cutoff2 / nyq
```

```
            b, a = butter(order, normal_cutoff, btype='high',
analog=False)
            y = lfilter(b, a, y)

            freq=int(numpy.fabs(int(numpy.floor(y.sum()))-138000))
            print(freq)

            if(freq>2000):
                    winsound.Beep(440, 300)


box = MyOVBox()
```

Prilog 3

KOD ZA ISCRTAVANJE FREKVENTNOG ODZIVA KORIŠTENOG FILTRA

```
import matplotlib.pyplot as plt
from scipy import signal
import numpy as np

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111)

b, a = signal.butter(4, 30, 'low', analog=True)
w, h = signal.freqs(b, a)
b2, a2 = signal.butter(4, 13, 'high', analog=True)
w2, h2 = signal.freqs(b2, a2)
plt.plot(w, 20 * np.log10(abs(h)))
plt.plot(w2, 20 * np.log10(abs(h2)))

plt.xscale('log')
plt.title('Butterworth filter frequency response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.axvline(30, color='green') # cutoff frequency
plt.axvline(13, color='green')

fig.savefig("graph.png")
```