TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tanel Ottis 164817

EEG LAINETE RAKENDUSE FUNKTSIONAALSUSE LAIENDAMINE

Bakalaureusetöö

Juhendaja: Eduard Petlenkov
Professor

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Tanel Ottis

18.05.2020

Annotatsioon

Antud lõputöö eesmärk oli viimistleda aluseks võetud tarkvara ning valmistada töötav rakendus, mis võimaldaks kasutajal katseisikul monitoorida ajukoore elektrilaineid, kasutades selleks MyndPlay EEG seadet. Andmeedastusena kasutatakse juhtmevaba Bluetooth ühendust, mis annab mobiilse vabaduse. Tulemus on kokku pandud üheks programmi failiks.

Lõputöös seletatakse lahti lahenduse eripärad ning võimalused edasiliikumiseks. Põhjendatakse, miks on Python keelel põhinev tulem lihtsalt täiendatav ning kuidas võiks ideaalne lõpp produkt välja kujuneda.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 10 peatükki ja 4 pilti.

Abstract

Functionality Extension of EEG Waveform Application

The aim of this thesis is to further develop an EEG waveform application by extending the back end functionalities. Final outcome allows the user to monitor a test subjects brain activity in real time via the user interface. Data transfer was implemented using Bluetooth connectivity for mobility. End result is conveniently packaged in a single program file.

The properties will be explained in the thesis aswell as possible future improvements and extensions to achieve the ideal application. It will be brought up why Python language was chosen and how program was written with the help of the MyndPlay Research Kit.

The thesis is in Estonian and contains 27 pages of text, 10 chapters and 4 pictures.

Lühendite ja mõistete sõnastik

EEG	Elektroentsefalograafia, peaaju aktiivsuse mõõtmise meetod
API	*Application Program Interface* ehk Rakendusliides

Sisukord

Piltide loetelu

1 Sissejuhatus

Käesoleva bakalaureusetöö raames keskendutakse ajusignaale mõõtva peapaela (edaspidi **Seade**) tarkvaralise rakenduse funktsionaalsuste täiendamistega. Seade on ühe kanaliga ning kolme kontaktmõõtepunktiga, mis toodetud firma MyndPlay poolt ning saadud juhendajalt, professor Eduard Petlenkovilt. Andmeedastuse meetodiks on juhtmevaba ühendus ehk Bluetooth. Resultaat valmis koos kolleegi Richard Ploompuuga. Ülesanded on kahe vahel ära jaotatud, millest kasutajaliidese disainimise ja loomise eest (ingl. keeles *front-end*) kannab hoolt Richard Ploompuu ning programmi otstarvekuse laiendamisega (ingl. keeles *back-end*) tegeleb Tanel Ottis.

Arenduse aluseks on võetud möödunud semestri (2019 a. sügissemester) õppeaine raames valminud lahendus. On kasutatud ka Seadme tootja poolt MATLAB keeles kirjutatud uurimuslikku komplekti, milles seisneb täpsemini lahti seletatud riistvara tööpõhimõte, milleta oleks staabiilne ühendus pea võimatu. On tarvilik teada kindlaid baidilisi väärtuseid korrektselt andmete tõlkimise saavutamiseks. Arendamise keeleks on võetud Python versioon 3.7, mida kasutavad nii kasutajaliides, kui ka taustal töötav funktsionaalsus.

Lõputöö eesmärgiks on seatud reaaalajas Seadmelt andmete kättesaamine ning töötlemine, sealjuures vastava signaali graafikule kuvamine. Tähelepanelik tuli olla piisavalt optimaalse programmikoodi kirjutamisel, võttes arvesse, et Python pole kõige kiiremini töötavam (võrreldes näiteks C-ga). Samas polnud ka aksepteeritav aeglase või liigselt ressursse kulutava lahenduse kasutamine.

Lõpptulemusena valmis soovitud programm, mis loob reaalajas Seadmega andmeedastuse, töötleb baitide väärtuseid vastavalt riistvaralisele spetsifikatsioonile ning lõpuks tekitab graafilise signaali pilid ka ekraanile lõppkasutaja jaoks. Programmikood leiab aset GIT keskkonnas Bitbucket.

2 Eesmärk

2.1 Probleem

Töös kasutatavale peapaelale ei eksisteeri veel sobivat monitooringu tarkvara. Elektrilained suudab küll riistvara ära lugeda, kuid lõppkasutajale on tulemus tühine, kui pole vastavat liidest, mis andmeid kuvaks. Kasutatav seade on võrdlemisi odav võrreldes võimekamate EEG tehnikaga, mis teeks annaks taskukohasema lahenduse. Pärast rakenduse valmimist on eesmärk hakata monitoorima katseisikut reaalses ja virtuaalses keskkonnas ning kuidas tema ajulained sellest tingitud võivad olla. Leidmaks, kas alateadvuses toimub silmale märkamatut ning kui kahjulik võib virtuaalmaailm olla.

2.2 Lahendus

Aluseks võetud tarkvara abil arendada välja kasutatav programm, mis suuteline EEG laineid mõõtvalt seadmelt Bluetooth ühendusel andmeid töötlema ning signaalväärtustest ilmutama kasutajaliidesesse reaalaja graafiku. Kaasates ka **uurimuslikku MATLAB komplekti [1]**, mis seadme tootja poolt väljastatud, on võimalik ka mõista andmeedastusprotokolli, leiutades spetsiaalse rakenduse käesolevale riistvarale. Lõpptulemuse programmikood jäetakse avatuks, kuhu saab tulevikus funktsionaalsust lisada ning isegi lainedada sobivust teiste sarnaste EEG seadmetega.

3 Aluseks võetud lahendus

Möödunud semestril (2019.a sügissemester) sai õppeaines „**Arvutite ja süsteemide projekt**“ aine koodiga **IAS1420** tehtud esialgne versioon. Ülesanne seisnes professor Eduard Petlenkovi poolt püstitatud kirjelduses, mille järgi oli vaja valmis saada rakendus, mis suudaks Seadmelt andmeid lugeda ning ka mugavalt kasutaja ekraanile kujutada saadud signaali. Resultaadis esines visualiseeritud signaal ning ka videopilt katseisikust, mis sai lindistatud katse tegemise ajal. Signaali väärtused salvestati tekstifaili läbi kolmanda osapoole tarkvara „**NeuroView**“ **[2]**, kuna oli veel teadmata andmeside toimimise protokoll. Ei suudetud reaalajas infot töödelda, mistõttu valiti kergem, kuid reaalne lahenduskäik. Pärast katset käivitati loodud rakendus, mis tekitas andmetest signaalipildi ühes ning videoklipp ilmus teises aknas.

Lahenduse eesmärk oli katseisiku jälgimine ning samaaegselt aju elektrilainete monitoorimine. Kasutaja on selliselt suuteline jälgima käitumist, reaktsioone, näoilmeid ja muud ning kuidas isiku füüsiline olek mõjutab vaimset seisundit. Käesolevas bakalaureusetöös on lahti seletatud osa võetud aluseks uue ja etema tulemuse saavutamiseks.

3.1 Signaalpilt

```
def use_pyqtgraph():
    app = qt.QtGui.QApplication([])  # Initialize

    win = pg.GraphicsWindow(title="Signal from buffer")  # creates a window
    # creates empty space for the plot in the window
    p = win.addPlot(title="Realtime plot")
    # create an empty "plot" (a curve to plot)
    curve = p.plot()

    windowWidth = 512  # width of the window displaying the curve
    # create array that will contain the relevant time series
    Xm = np.linspace(0, 0, windowWidth)
    ptr = 0  # set first x position

    stream_file = open('test_files/test_stream/stream1.txt', 'r')

    # read line (single value) from the serial port
    value = read_from_stream(stream_file)
    for val in value:
        # shift data in the temporal mean 1 sample left
        Xm[:-1] = Xm[1:]
        try:
            # shift data in the temporal mean 1 sample left
            Xm[-1] = float(val)
        except ValueError:
            pass
        ptr += 1  # update x position for displaying the curve
        curve.setData(Xm)  # set the curve with this data
        curve.setPos(ptr, 0)  # set x position in the graph to 0
        qt.QtGui.QApplication.processEvents()  # process plot

    # END QtApp #
    pg.QtGui.QApplication.exec_()
```

Pilt 1 Pyqtgraph lahendus

Järgmisena lahti seletatav **lahendus [3]** (vt. Pilt 1 Pyqtgraph lahendus) kasutab eelnevalt salvestatud tekstifaili väärtuseid taasimiteerimaks katse käigus tekkinud signaalipilti ning on leitav failist **myndplay/other/other_functions.py** failst. Teegi **pyqtgraph** abil realiseeritud funktsiooni **use_pyqtgraph** tulemus on väljanägemiselt erinev antud lõputöö omast. Lahendus seisnes esmalt tühja graafiku akna tekitamises. Määrati ära maksimaalne X-telje laius *windowWidth* ning võeti appi **numpy** kogumik, mille abil tekitati nullidest koosnev signaal *Xm*. Algväärtustati nullpunkt ning loeti fail sisse kasutades **read_from_stream** funktsiooni. Seejärel hakati ilmutama graafikut nihutades iga tsükliringiga X-telje nullpunkti ning uuendades signaali. Tulemus andis küll soovitu, kuid kulutades selle käigus absurdselt 30-40% protsessori jõudlust.

3.2 Videotöötlus

```
class FileVideoStream:
    def __init__(self, path, queue_size=128):
        # Initialize the file video stream along with the boolean
        # Used to indicate if the thread should stop or not
        self.stream = cv2.VideoCapture(path)
        self.stream.set(cv2.CAP_PROP_BUFFERSIZE, 32)
        self.stopped = False

        self.FPS_MS = int( (1/30) * 1000 )

        # Initialize the queue used to store frames read from the video file
        # self.Q = Queue(maxsize=queue_size)
        self.frame = None

        # Start a thread to read frames from the file video stream
        self.thread = Thread(target=self.update)
        self.thread.daemon = True
        self.thread.start()

    def update(self):
        # Keep looping forever
        while True:
            # If thread indicator variable is set, stop the thread
            if self.stopped:
                return

            # Read the next frame from the file
            grabbed, self.frame = self.stream.read()

            # If grabbed == False, then video is at end
            if not grabbed:
                self.stop()
                return

    def read(self):
        # Return next frame from queue
        # frame = self.Q.get()
        cv2.imshow("Frame", self.frame)
        # Hoiab pilti korraks
        cv2.waitKey(self.FPS_MS)
        # return self.Q.get()

    def stop(self):
        # Indicate that the thread should be stopped
        self.stopped = True
```

Pilt 2 FileVideoStream klass

Põhiliseks teguriks on klass **FileVideoStream** (vt. Pilt 2 FileVideoStream klass), mis põhineb **OpenCV lahendusel [4]**. Initsialiseerides avatakse salvestatud videofail etteantud puhvrisuurusega. Määratakse ka konstant, mida hiljem tarvis, et tekiks pilt ~30 kaadriga sekundis ning mitte rohkem. Vastasel juhul hakkab protsess tööle nii kiirelt, kui vähegi suudab, millega kaasneb meeletu ressursikulu. Luuakse sekundaarne lõim, mis hakkab tööle **update** funktsiooniga. Lõppematus tsüklis kontrollitakse ega protsess lõppenud pole. Hakatakse haarama üksteise järel kaadreid videofailist ning salvestama **frame** muutujasse. Pealõim jätkab tööd käivitades **start_video_feed** funktsiooni, milles järjekordne tsükkel kuvamisülesannet täidab. Funktsioon **read** ilmutab hetkeks kaadri ning teeb seda, kuni nende lõppemiseni.

3.3 Tekkinud murekohad

Probleeme esines eelmainitud süsteemis mitmeid. Nimelt andmetöötlus ning video- ja signaalipidli kuvamine kurnasid arvutiprotsessorit 70-80% ulatuses. Lihtsa programmi jaoks on seda liiast. **opencv-python** teegist pärinev paketi **cv2** klass **VideoCapture** oli peaelemendiks, läbi, mille **read** funktsiooni suudeti järgmisi kaardeid lugema. Leides, et sealtsamast ka kurjajuur pärineb. Nähti ränka vaeva uute piltide kättesaamiseks ning nende ilmutamiseks. Läheneti videotöötluse suunas, mis ongi väga resursikulukas. Saladus seisnes asjaolus, kuidas üldse filmiklippi salvestatakse ning lahti dekodeeritakse. Protsess on väga arvutusintensiivne (ingl. keeles *compute-intensive*) ja spetsiifilisemalt mõeldud hoopis graafikakaardi ülesandeks, kuna too suudab arvutusi teostada efektiivsemalt, kui protsessor. Algoritmi edukaks refaktoriseerimiseks oleks kulunud liialt aega ning poldud suutelised enam projekti raames teostama.

Lõppkasutajale kuvatavad aknad olid omavahel lahutatud, justkui kaks eraldiseisvat. Viisakamaks väljanägemiseks oleks võinud ka programmi ennast kasutajal lihtsalt käivitada võinud olla, kuid programmikoodist viimistlemistega kaugemale ei jõutud.

Praegune bakalaureusetöö keskendub ka eelnevalt nimetatud probleemide eemaldamisele. Valmistatud programmi on lõppkasutaja iseseisvalt suuteline käivitama ning eelneva 70-80% koormuse asemel on rakendust optimiseeritud ning jõutud ~10% kasutuseni. Lahti selgitatud lahendus leiab aset **myndplay/other** kaustas, mida lõpptulemuses ei kasutata. Jäetud alles vaid esialgse versiooni näidisena.

4 EEG ehk elektroentsefalograafia

Inimeste üheks tähtsaimaks ning kõige väärtuslikumaks organiks on aju. Elund, mille loomus on imeline, kuid samas meeletult keeruline, et siiamaani seda jätkuvalt uuritakse. Mõttetegevuse toimimiseks liiguvad neuronite vahel ringi saadetud väikesed energialaengud. Elektroentsefalograafia on nende põhjal **ajukoore elektrilise aktiivsuse uurimise ja monitoorimise meetod [5]**, mis annab informatsiooni ajukoore seisundi ja funktsioneerimise kohta. Meditsiinis kasutatakse EEG uuringuid leidmaks erinevate ajuga seotud haiguste, nagu näiteks epilepsia, teadvushäired, depressioon, diagnoosimiseks. Teatud kordadel on ka võimalik uuringu tulemusest välja lugeda määratud ravi efektiivsust.

Käesolevas bakalauresetöös on lõpptulemusena valminud rakendust soov kasutada ajutegevuse erinevuste leidmiseks reaalelus ning virtuaalses keskkonnas oleval isikul. Alles 2010. aastast hoogu kogunud VR-peakomplektid on huvitavad, kuid nendega seotud võimalikke pikaajalisi kõrvalmõjusid ei osata veel korrektselt hinnata.

5 Riistvara ehk MyndPlay EEG peapael

Pilt 3 MyndPlay EEG seade

Rakenduse tuumana töötab aju elektrilisi signaale ehk **EEG laineid mõõtev seade [6]**. Jälgimispunktiks on inimese pea otsmiku keskel asuv punkt. Optimaalseimaks töötamiseks on riistvarale tehtud mugav peapael, mis paigutatakse katseisikule pähe. Sellisel kujul kandes ei takistata erinevaid tegevusi (näiteks jooksmine või pallimäng). Põhiosadeks on kolm komponenti, milleks on mõõteriba, Bluetooth üksus ning tekstiilist peapael (vt. Pilt 3 MyndPlay EEG seade). Korrektseks toimimiseks on vaja saavutada juhtmevaba ühendus arvutiga. Seadmel on küljes sisse- ja väljalülitamise nupp. Erinevates faasides hakkab nupp helendama kindlat tooni valgust, andes sellega teada tööolekust.

Juhtmevabaks töötamiseks on kaasas ka väike liitiumioonaku, millel toidet kuni kümneks tunniks. Eelnevalt mainitud on Seadmel üks kanal ja kolm sensorit. Mõõteriba registreerib otsmikult tuvastatud elektrisignaalid ning saadab need analoogsel kujul edasi Bluetooth üksusele. Seejärel töödeldakse väärtused digitaalsele kujule ja saadetakse ühendatud arvutisse.

Väljundandmeid on mitut eri sorti. Põhilisem neist on toores ajulaine signaal maksimaalse mõõtesagedusega 512 Hz. Lisaks saadetakse teatud ajavahemiku tagant ühenduvuse tugevuse analüüsi tulemus, mille põhjal saab juhtmevaba ühendust korrigeerida. EEG võimsusspektri (ingl. keeles *power spectrum*) lugemine on samuti võimalik. Praeguse lõputöö raames keskendutakse konkreetsemalt toore signaali töötlusele, kuna mitme erineva väljundandme jaoks sobiva rakenduse arendamine juba ületaks bakalaureusetöö mahtu.

6 Kasutatud ressursid

Töös jälgiti Seadme tootja poolt väljastatud abistavat MATLAB keeles kirjutatud komplekti. Parima kokkusobivuse huvides sai seetõttu valitud ka arendamiskeeleks Python 3.7.7, millel olemas ka teek nimega „**matplotlib**“, mis võimalda just MATLAB-i ja Python-i vahelist koostööd. Kaks põhilist protsessi peavad paralleelselt jooksma, seetõttu on võetud kasutusele lõimed (ingl. keeles *threading*), tänu millele suudavad protsessori loogilised tuumad tegeleda erinevate tegevustega samaaegselt. Bluetoothi ühenduse saavutamiseks rakendati teeki „**pySerial**“, mis vastavaks eesmärgiks loodud. Arenduskeskkonnaks valiti PyCharm Community Edition.

6.1 Python 3.7.7

Programmeerimiskeelena sai valitud Pyhton 3.7 versioon. Antud bakalaureusetöö valmimise hetkel viimane väljalase oli 3.7.7. Loojad tahtsid jätkata eelnevalt tehtud osa sama põhja pealt, mistõttu ei oleks tarviklik hakata refaktoriseerima. Sooviti lahenduses lihtsust ja agiilset alust ajakulu kokkuhoiu mõttes. Interpreteeritava keelena ei ole **pyhton kõige kiirem [7]**, kuid kasutuskergus ning kiire arenduskäik soosisid olukorda. Samuti vabavarana kasutatavaid teeke ning erinevaid lahenduskäike esineb meeletult, tänu millele on ka eesmärgi saavutamine hõlbustatud. Valmis saadud tulemuselt on võimalik arendust muuta optimaalsemaks näiteks, kui programmikood tõlkida ümber kurikuulsasse C-sse või hiljuti leiutatud Go keelde. Tulemusena paraneks veelgi rakenduse töökiirus ning ressursikulu vähenemine. Lahenduse aga nullist selliselt realiseerimine nõuaks rohkem aega ning põhjalikumat süvenemist. Nüüdseks, kui algkuju juba loodud, on konkreetsem alus olemas.

6.2 Teek pySerial

Põhiline täidesaatev **moodul**, mis vastutab juhtmevaba andmeside eest. Tekitab **seriaalühenduse [8]**, konfigureerides Seadme serveriks ning vastuvõtva poole ehk arvuti kliendiks. Sügavamal masinalähedasem suhtlus valitakse automaatselt teegi enda poolt. Omab sama klassi liidest erinevatel operatsioonisüsteemidel. Läbi pordi luuakse binaarne ülekanne (ingl. keeles *transmission*), mis muudab mooduli universaalselt kasutatavaks.

6.3 Teek matplotlib

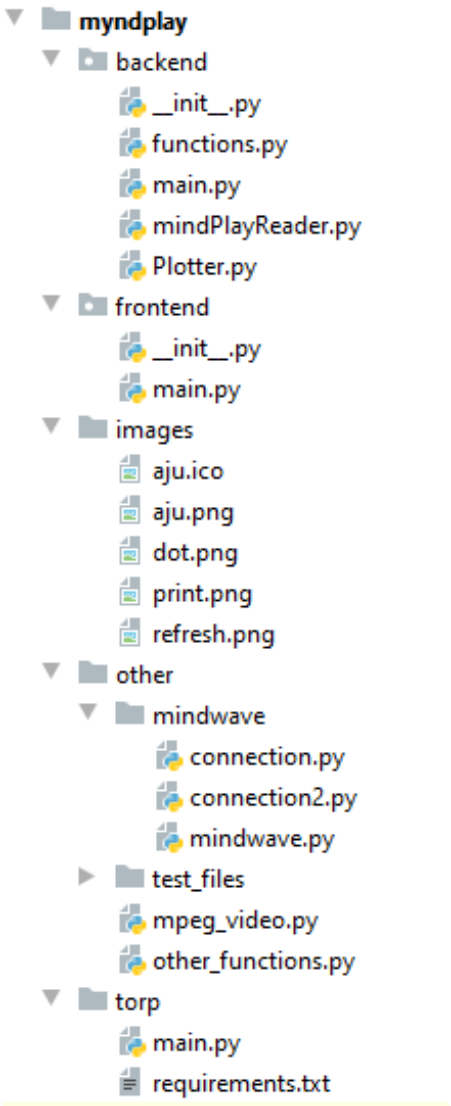
Üks populaarseimatest ja enimkasutatavatest lahendustest vektoriaalsete graafikute kujutamisel. Rakenduse kasutajaliidese kasutab graafilise liidese tööriistakomplekti **Tkinter**, millega matplotlib ühtib läbi enda **objektorienteeritud API [9]**. Teek on disainitud MATLAB-i lahendusi realiseerimiseks python rakendustes, olles samuti vabavarana kasutatav. Lõpptulemuses implementeeritud **pyplot** moodul aitab tekitada ka MATLAB-le tuttavaid liideseid. Funktsionaalsused hõlmavad sinusoidi laadse graafiku kuvamine, histogrammi, 3D-kujutise ja veel paljude teiste ilmutamine.

6.4 Lõimed ehk *threading*

Programmeerimises tüüpiliselt täidetakse käske mööda ühte jada, mis arvutile ette kirjutatud. Vahel võib tekkida vajadus täita mitut funktsiooni samaaegselt ehk teostada paralleeltöötlust. Eeldatakse, et protsess võib koosneda hulgast erinevatest **striimidest, mis vajavad üheaegset töötlemist [10]**. Mitmelõimelise olukorra puhul tekitatakse ajapõhine jaotus, mille vältel protsessor erinevate lõimede vahel ümber lülitub. Inimesele tekib illusioon samaaegsusest, kuna ajavahemik on niivõrd väike. Mitmetuumalistes või -protsessorilistes süsteemides saab sama tulemuse kasutades multitöötlust. Sellises olukorras ei pea ilmtingimata üks tuum pidevalt ennast lõimede vahel jagama, vaid kasutatakse ära teisi, mis töötlust kiirendab.

Lõimi saab jaotada vastavalt prioriteedile, mis määrab ära teatud protsessi teostuse tähtsusastme teiste lõimede suhtes. Sellise omadusega hakkavad käivituma esmatähtsad funktsioonid esimesena ning madalama prioriteediga lõimed alles, kui riistvara seda võimaldab. Täpsem realiseerimine hakkab olenema juba operatsioonisüsteemi ning arhitektuuri spetsiifikast.

7 Realiseeritud lahendus



Pilt 4 Failide struktuur

Pildil (vt. Pilt 4 Failide struktuur) on ära näidatud failide struktuur. Järgnev lahendus asub **myndplay/backend** kaustas. Lisadena juures olevad teegid asuvad virtuaalkeskkonna kaustas **myndplay/venv**, mis on ühises kasutuses ka Richard Ploompuu tehtud tööga.

7.1 Töö käik

Lõpptulemus tekib kahe protsessi paraleelsel tööl. Neid saab eristada, kui arvutuslikku ning visualiseerimist. Nimelt arvutuslik pool tegeleb Seadmelt saadud andmete töötlemisega ja korrektsete signaaliväärtuste salvestamisega. Teine külg kasutab eelnevalt registreeritud andmeid ning jooksvalt koostab kasutaja ekraanile graafiku, mis pärineb katseisiku ajulainetest. Tuleb tähele panna, et korrektse toimingu nimel peab esmalt alustama tööd arvutuslik osa. Vastasel juhul püüakse ilmutada graafikut olematute väärtustega, mille tulemusel tekib viga. On tegemist reaalajas signaali monitoorimisega, mistõttu ka kuvatud graafik on pidevas muutumises. Põhjus selliseks paralleeltöötlemiseks on vähendada peiteaega (ingl. keeles *latency*) ning muuta rakenduse käitumist sujuvamaks.

7.2 Arvutuslik pool

Protsess käik saab alguse **main.py** failist. Esmalt tuuakse sisse **mindPlayReader.py** failist leiduv klass nimega **MyndPlayReader**, mis on põhilisemaks objektiks arvutuslikus protsessis. Tekitatakse sõnastik (ingl. keeles *dictionary*) võimalikest COM portidest. Peale seda algväärtustatakse **MyndPlayReader** klass sobiva pordi numbriga, mille kaudu hakkab toimuma juhtmevaba andmevahetus. Initsialiseerimise käigus luuakse vastava klassi instants koos sobilike parameetritega ning saavutatakse Bluetooth ühendus. Vea esinemisel lõpetatakse protsess ning kuvatakse kasutajale veateade. Defineeritakse ära funktsioon **start_mind_reader**, mis paneb ühe protsessori loogilise tuuma alustama andmete lugemist ning käivitatakse **MyndPlayReader** klassi funktsioon **run**.

Esmalt kontrollitakse, stabiilset ühendust parameetriga **closed**, milles tõene väärtus viitab juba lõppenud protsessile. Korrektsel toimimisel pöördutakse funktsiooni **read_serial** poole, mis tagastab andepaketid ning tooreid baidi väärtused. Teeki **pySerial** on implementeeritud puhver andmetega, mis Seadme poolt välja saadetud, kuid mitte veel rakenduse poolt vastu võetud. Oodatakse kuniks puhver täitub, kontrollides vastavat **pySerial** instantsi **in_waiting** omadust. Nullist suurema väärtuse puhul on tegemist ootavate baitidega ning protsessi saab jätkata. Seejärel loetakse sisse kogu sisu muutujasse **raw_buffer**. Juhul, kui baite oli rohkem kui 2048, siis on võimalik, et osa andmeid läks kaduma ületades puhvri maksimaalset suurust. Toored andmed lisatakse klassi muutuja **total** sisse, mis hoolitseb poolikute pakettide (ingl. keeles *packet*) leidmisel. Järgmisena otsitakse „python regex“-i ehk regulaarse väljendi abil järjestikku olevate baitide asukohad ehk indeksid, mille väärtused on 0xAA, mis kujutab endast uue paketi algust. Kindluse mõttes on imporditud failist **functions.py** funktsioon **check_indexes**, millega kontrollitakse, ega teatud 0xAA baidid ei olnud reaalsed andmed, näiteks, kui esineks kolm tükki kõrvuti. Jäetakse järele vaid kõige viimaste baitide paar. Leitud indekseid peab olema üle kahe. Vastasel juhul on tegemist pooliku paketiga ning tulemus lisatakse **total** muutujasse, kus juba on või alles tulemas ülejäänud poolik osa. Nüüdseks on andmed Seadmelt omandatud ning vaja asuda töötlemise poole.

Kätte saadud teave edastatakse funktsioonile **process_serial**, milles eraldatakse baidis olevad signaali väärtused. Veendutakse, et paketi pikkus peab olema vähemalt kahe baidi suurune. Liigutakse edasi paketi verifitseerimise juurde funktsioonis **verify_packet**. Esimese sammuna saadakse reaalsete andmetega (ingl. keeles *actual data*) baidid ning leitakse koormussumma. Selle leidmiseks teostatakse biti tasemele JA-tehe andmebaitide ning 0xFF (maksimaalne baidi summa) väärtusega. Sama tehtega saadakse ka kontrollsumma, kasutades paketi viimast baiti, kuid lõpuks inverteeritakse kogu saadud tulemus. Tagastavas väärtustes on baidist reaalsed andmed ning võrdlustulemus kontrollsumma ja koormussumma vahel, kus tõene viitab andmete korrektsusele. Peale paketi sobivuse kontrolli veendutakse kätte saadud kahendväärtuses ning jätkatakse parsimisega, mida teostab **parser** funktsioon.

Hakkab toimuma tsükkel, milles iga ringiga uuritakse järgmist baiti individuaalselt. Riistvaraliselt on protokolli sisse seatud nn. *EXCODE* väärtused (0x55), mille tähenduses käesoleva lõputöö koostaja ei ole päris kindel. Uurimuslikust MATLAB-i komplektist on aru saadud, et selliste baitide eksisteerimisel, eelnevad need signaaliinfole. Vahetult peale viimast *EXCODE* baiti esineb *CODE* väärtus, mis kujutab endast signaalibaitide kogust. Väiksem kui 0x80 väärtus vihjab ühele ning suurem või võrdne kui 0x80 mitmele. Vastavalt salvestatakse otsitavad väärtused ning tagastatakse **process_serial**-i, mis ka jõuavad kohe tagasi **read_serial**-i ning lõpuks **run** funktsiooni. Olematute ehk NULL või None väärtusega andmete kättesaamisel korratakse siiani toimunud protsessi.

On jõutud kohta, kus omistatud kõik puhastatud baidid sisendpuhvrist ning võimalik edasi liikuda konkreetsete graafikul väärtuste eraldamiseni. Järjekorras olevate elementide esimeseks baidiks on sama CODE väärtus, vastavalt millele käivitatakse ka funktsioon **get_multibyte_raw_value** või **get_singlebyte_raw_value**, mis eraldab ühe või mitu baiti. Järgnevalt omistatakse saadud X-telje tulemus **x_axis** klassi muutujasse, mida hakkab graafiku projekteerimise osa kasutama. Y-telje muutujale **y_axis** lisatakse juurde element suurusega $n + 1$, kus n on eelmise liikme väärtus. n_0 algväärtustatakse nulliga.

7.2.1 Lahenduse omapärad

Erinevalt tavapärasest python-i rakendustest on töös asendatud teadatuntud nimekiri ehk **list** (teise nimega „**array**“) asendatud teegist **collections** pärit **deque** (ingl. keeles *double ended queue*) elementidega. Tegemist on reaalaja rakendusega, mistõttu soovib lõputöö koostaja ka minimaalselt ressursse kulutada ning protsessi võimalikult palju kiirendada. Uus **deque** objekt suudab teostada vajaolevaid eesmärke samamoodi nagu **list**, kuid on omajagu optimaalsem oma teostuses. Nimelt on objekt loodud sihiga just andmete juurdepääsuga otspunktides. Loodud operatsiooni käigus toimub pidev **x_axis** ja **y_axis** muutujatest väärtuste vasakult poolt elemendi eemaldamine käsuga **popleft()** ning paremale juurde liitmine operatisooniga **append()**.

Loodud on sõnastik **switcher**, mis täidab *switch-case* rolli. Python-is ei ole sellist talituslikkust loodud ning nõuab loovat lahenemist. Objekt koosneb võtmetest ning nende poolt käivitavatest funktsioonidest. Kindla väärtuse puhul saadakse võtmeväärtusele omane funktsioon, mida käivitada. Selline käitumine eemaldab mitmete *if-else* lausete lisamise, millega kaasneks protsessi aeglustumine ning koodistiili kehvenemine.

7.3 Graafiline pool

Protsess algab **main.py** failist. Lisaks veel eelnevalt mainitud arvutuslikule osale imporditakse **Plotter** failist klass nimega **Plotter** ning initsialiseeritakse graafik, kasutades teegi **matplotlib**-i figuuri (ingl. keeles *figure*). Samuti luuakse kindla vahemikuga teljestikud. Alles pärast **start_mind_reader** käivitamist pöördutakse **start_plotting** funktsiooni poole. Samamoodi tekitatakse uus lõim (ingl. keeles *thread*) signaali projekteerimiseks. Käivitatakse **run_time_plotting** andes kaasa parameetrina **MyndPlayReader** instantsi, mille kaudu pääsetakse ligi X ja Y-telje väärtustele.

Käsuga **plot()** tekitatakse teljestik-tüüpi muutuja. Enne tsükli alustamist kontrollitakse, kas signaalväärtused on juba arvutusliku poole läbinud ehk teljestiku punktid ei oleks initsialiseeritud andmetega ehk nullid (arv 0). Süsteem ootab, kuniks baite töödeldakse. Kasutades alamteegist **matplotlib.animation** funktsiooni **FuncAnimation** parameetriga **blit** tõese kahendväärtusega hoitakse kokku ressursikuludelt, kui ka kuvamiskiiruse osa pealt. Tüüpiliselt tekitataks graafikupilt staatiline ning reaalaja aplikatsiooniks oleks tarvilik pidevalt vana illustratsioon kustutada ning uus asemele ilmutada. Selline protsess kulutab, aga palju jõudu, kuna üks raskemaid ülesandeid arvutil on joonistuse ilmutamine ekraanile. Praegusel lahendusel, aga lähenetud teise külje alt. Eelmainitud parameeter muudab signaalijoonist ainult uuendades selle X ja Y-telje väärtuseid ning laseb seejärel akent värskendada. Hoitakse kokku tehtud töö pealt, kuna ei ole vaja pidevalt graafikut kinni panna ning kohe uus tekitada.

Realismi ja optimaalsuse huvides on seatud värskenduskiiruseks kord iga 34 millisekundi jooksu. Kasutaja hakkab nägema ~29 kaardit sekundi kohta, mis on sarnane televisioonis nähtava filmiga. Protsessi initsialiseerib **init_animation** tühjade väärtustega teljestikul. Funktsioon **animation** täidab primaarset eesmärki, milles iga värskendustsükkliga omistatakse Seadmelt saadud X ja Y andmed kuvatavale graafikule. Samuti nihutatakse X-teljestikku ühe võrra edasi, kuna toimub ajas liikumine.

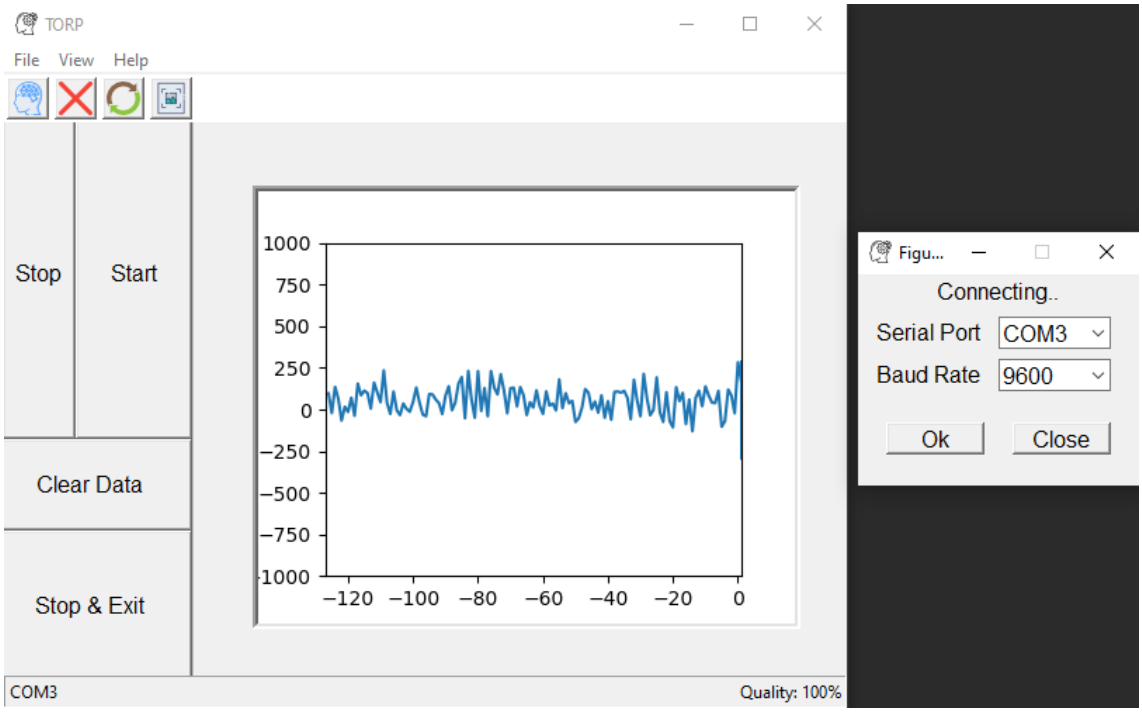
7.3.1 Lahenduse omapärad

Kasutatud on pyhton-i üht kõige tüüpilisemat graafiku kuvamismeetodit. Lahenduses kasutatakse **matplotlib** teegi **pyplot** moodulit. Soovitud eesmärgi saavutamiseks on selline teguviis *de facto* standard. Kahjuks pole tulemus kõige optimaalsem. Efektiivsust suudetaks parandada näiteks **pyqtgraph** teegiga kaasnevat **QtGui** funktsionaalsust. Käesolevas töös oli tähtis protsessi ilmtingimata mitte liialt keerukaks ajada ning keskenduti varem tuntud ja testitud variandile. Sellegipoolest jõutakse viisaka lõppproduktini, kasutades abistavat **FuncAnimation** klassi, mis täidab vajalikke nõudeid.

Jooksva signaalipildi tekitamiseks on tõene väärtus omistatud **blit** parameetrile, mille peale hakatakse kasutama ***blitting* töövõtet [11]**. Termini, mis on tihtipeale tuntud arvutigraafika maailmas, sisu seisneb blittimisalgoritmile muutumist vajava graafikuosa värskendamises. Selliselt hoitakse ressursse kokku ajutiselt paigal püsiva signaaliväärtuse pealt. Vastasel juhul kustutataks seisund terviklikult ning ilmutataks koheselt uus.

EEG aktiivsus on võrdlemisi madal ning mõõteühikuks on mikrovolt. Rakenduses on mõõtevahemikuks seatud 1000 µV kuni -1000 µV ja seda sellepärast, et tavapäraselt etteantud vahemikust väljaspool esinevat võimsust ei eksisteeri tavapärase inimese monitoorimisel. Kasutatud on lihtsa madal- ja kõrgpääsfiltri põhimõtteid vigade eemaldamiseks.

8 Kasutajaliides



Pilt 5 Kasutajaliides

Ekraanile ilmuvas kasutajaliideses on kõik vajaminev rakenduse kasutamiseks. Vastavalt pildile (vt. Pilt 5 Kasutajaliides) on ka realiseeritud nuppude funktsionaalsused. Esmakordselt programmi avades kuvatakse väiksem aken (pildil väiksem aken paremal), mille kaudu saadakse valida COM port ning ka ühenduskiirus (ingl. keeles *baud rate*) ning läheb vaja süsteemi korrektsel toimimisel. Korrektsed parameetrid sisestatud saab vajutada nuppu „Ok“, mis käivitab jooksva signaali graafiku. Nupu „Stop“ kaudu saab graafikut seisma panna, juhul, kui esineb soov põhjalikumat signaali uurida ning „Start“ abiga taas pausilt maha võtta. „Clear Data“ puhastab kogu graafikul oleva info, et saaks alustada uuelt puhtalt lehelt. „Stop & Exit“ täidab programmist väljumise eesmärgi, kui katse monitoorimine ei ole enam vajalik.

Lahendus on leitav **frontend/main.py** failist, läbi mille pääseb põhiprotsess ligi ka tagatausta funktsionaalsustele.

9 Järgmised sammud

Realiseeritud lahendus täidab põhifunktsioone, kuid võttes arvesse, mida Seade teha suudab, on veel arengumaad. Nimelt toore signaali kuvamine sai tehtud, millega ka käesolev bakalaureusetöö piirdub, kuid näiteks EEG võimsusspektri töötlemine ja kuvamine on tulevikku minev teema. Puudujäänud väljundite seas on veel keskendumis- ja mediteerimisvõime hinnang ning silmapingutuste tuvastamine võimalike mõõtevigade parandmiseks. Loetletud funktsionaalsuste täideviimine vajab lähenemist teistsuguse külje alt, kuna poleks enam tegemist ainult signaaligraafikuga. Kindlasti tarvis tutvuda põhjalikumalt uurimuskomplektiga, milles vajalik info ka olemas. Tuleks arvestada ajakuluga, mis läheb MATLAB skriptide analüüsimiseks, sest maht on suur ning python keelde implenteerimiseks pole üks-üheselt lahenduskäik alati tark.

Täienduste tegemisel on soovitatav EEG alal spetsialistiga läbi rääkida ning spetsiifilisemalt kokku leppida nõuetes, mida oleks tarvis ekraanipildil näha. Algne eesmärk oli siiski arendada rakendus lähtudes otse lõppkasutaja soovitud nõuetest. Lõputöö raames kohtuti Tallinna Tehnikaülikooli Tervisetehnoloogiate instituudi professori Maie Bachmanniga, kes jagas infot, et vaatamata erinevatele EEG monitoorimise eesmärgil loodud lahendustele ei ole alati tootjad kursis reaalelu vajadustele. Jätkutööna tuleks nimetatud aspekti silmas pidada.

Finaliseeritud toode oleks soovituslikult dünaamilise väljanägemisega, mis annaks kliendile vabaduse vastavalt enda mugavusele aknaid, signaalivärve või muud konfigureerida. Ressursikulu ei tohiks ületada tavalise sülearvuti riistvaralist võimekust, kuna soosiks katseisiku monitoorimist ka mobiilselt ehk liikumist takistamata. Hetkene valmis lahendus on leitav Bitbucket GIT-keskkonnast ning jäetakse **avatud lähtekoodiga kasutamiseks [12]**.

10 Kokkuvõte

Aju on kompleksseim organ inimkehas. Siiani tegeletakse selle täieliku mõistmisega, kuid õnneks tänu elektroentsefalograafiale oleme võimelised rohkem aru saama. Samal eesmärgil sai ka algatatud käesolev bakalaureusetöö. Võetud aluseks eelnevalt valmistatud projekt ning tegeleti protsessi viimistlemisega. Kasutades ajuaktiivsust mõõtvat seadet oli tarvis leituada rakendus, mis võimaldaks inimpea otsmikult uurinut monitoorida.

Tutvudes lähedasemalt EEG teemaga, sai võetud programmeerimskeeleks Python versioon 3.7, mis soosis agiilsemat arendust lihtsamalt teostada. Keele laia tugikonna ning multifunktsionaalsuse baasil loodi prototüüp laadi rakendus, mis korrektselt läbi juhtmevaba andmeühenduse peapaela mooduliga saavutas ning kasutajapoolseks jälgimiseks ka ekraanile signaalikujul ilmutas. Lõpptulemuses panustati koostööna võrdselt, mille puhul lõputöö koostaja Tanel Ottis vastutas funktsionaalsuste täiendamise eest ning tema koostööpartner Richard Ploompuu tegeles kasutajaliidese disainimise ja implementeerimisega. Riistvaraga kaasnenud uurimusliku komplekti abita oleks teostus olnud võimatu, kuna võtmeteguriks osutus suhtlusprotokoll. Andmebaitide töötlemine ning reaalsete väärtuste eraldamine vajas põhjalikku süvenemist. Ajuaktiivsust mõõtva seadme väljunditena said realiseeritud põhifunktsionaalsused, kuid sekundaarsete väljundite funktsionaalsused jäeti kõrvale, kuna kahjuks ei mahtunud enam töö raamesse.

Valmistatud tulem kavatsetakse jätta avatud lähtekoodiga kättesaadavaks, eesmärgiga tulevikus veelgi rakendust optimeerida ning täiustada vastavalt lõppkasutaja soovidele. Protsess vajab soovitud nõuete puhul arutlemist ja riistvara töövõime kõrgendamisel isegi terve komplekti ümbertegemist näiteks programmeermiskeelde C või Golang.

Kasutatud kirjandus

[1] „MyndBand + MRT Research Toolkit“ MyndPlay Ltd [Online] Saadaval: https://store.myndplay.com/products.php?prod=47 [Külastatud: 10-Okt-2019]

[2] „NeuroView 4.2“ Software Informer [Online] Saadaval: https://neuroview.software.informer.com/4.2/ [Külastatud: 16-Okt-2019]

[3] „Python Realtime Plot Using Pyqtgraph“ Stack Overflow [Online] Saadaval: https://stackoverflow.com/questions/45046239/python-realtime-plot-using-pyqtgraph [Külastatud: 03-Nov-2019]

[4] „Faster Video File FPS With cv2.VideoCapture and OpenCV“ Pyimagesearch [Online] Saadaval: https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/ [Külastatud: 29-Okt-2019]

[5] „Patsiendiinfo / Uuringud / Elektroentsefalograafia (EEG)“ Tartu Ülikooli Kliinikum [Online] Saadaval: https://www.kliinikum.ee/patsiendiinfo-andmebaas/elektroentsefalograafia-eeg/ [Külastatud: 03-Märts-2020]

[6] „MyndBand EEG Headset + MyndCap Bundle“ MyndPlay Ltd [Online] Saadaval : https://store.myndplay.com/products.php?prod=49&fbclid=IwAR2iEtBccl6-0RP2IC7EJsommw5X25U3tmPMM99AD8VAcpbUWfeb9day3zQ [Külastatud: 23-Sept-2019]

[7] „Why C Runs So Much Faster Than Python“ Huffpost [Online] Saadaval: https://www.huffpost.com/entry/computer-programming-languages-why-c-runs-so-much_b_59af8178e4b0c50640cd632e?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAFuENt7_zZDfxCBVpgGNWxsSepYkqClNNbXj0P47ffK9c-qxjX-OIAwbC1HhfSuyuNGoYy8_dSkm0J-MwgMqvRUJ0KYBe7Snt3RwcUdILzLNlUm2K63K5gm3E6QT6foByQrisCCQkTXDtr-gwHQNTuHr5Wkbh7C2m-aunsOygnkv [Külastatud: 03-Apr-2020]

[8] „pySerial’s documentation“ pySerial readthedocs [Online] Saadaval: https://pyserial.readthedocs.io/en/latest/ [Külastatud: 03-Veebr-2020]

[9] „Matplotlib“ Wikipedia [Online] Saadaval: https://en.wikipedia.org/wiki/Matplotlib [Külastatud: 01-Nov-2019]

[10] Lõimed: https://et.wikipedia.org/wiki/L%C3%B5im_(informaatika) [Külastatud: 18-Nov-2019]

[11] Blitting: https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.animation.FuncAnimation.html [Külastatud: 06-Märts-2020]

[12] Lähtekood: https://bitbucket.org/TaneelPaneel/myndplay/ [Külastatud: 29-Sept -2019]

Lisa – Arenduses kasutatud teegid

Sisu samuti leitav **requirements.txt** failist.

appdirs versioon 1.4.3
attrs versioon 19.3.0
black versioon 19.10b0
click versioon 7.1.1
cycler versioon 0.10.0
kiwisolver versioon 1.1.0
matplotlib versioon 3.2.1
numpy versioon 1.18.2
pathspec versioon 0.7.0
pyparsing versioon 2.4.6
PyQt5 versioon 5.14.1
PyQt5-sip versioon 12.7.1
pyqtgraph versioon 0.10.0
pyserial versioon 3.4
python-dateutil versioon 2.8.1
regex versioon 2020.2.20
six versioon 1.14.0
toml versioon 0.10.0
typed-ast versioon 1.4.1