СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра програмування та математики

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”

на тему „Розробка програмних модулів функціоналу інтерфейсу управління комп'ютером”

Виконав: студент групи ІПЗ-16д _____ Є. В. Воронін _____
(підпис) (ініціали і прізвище)

Керівник _____ В.Г. Іванов _____
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ В.О. Лифар _____
(підпис) (ініціали і прізвище)

Рецензент_____

Сєвєродонецьк – 2020

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра програмування та математики

Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”
(назва спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ,
д.т.н., доцент
_____ Лифар В.О.
“____” _____2020 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

_____ Воронін Євген Володимирович_____
(прізвище, ім’я, по батькові)
1.Тема роботи Розробка програмних модулів функціоналу інтерфейсу управління комп'ютером_____
_____
Керівник роботи_____к.т.н., доцент Іванов В.Г_____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “___”_____20__року №___

2. Строк подання студентом роботи_20 травня 2020 р._____ _____
3. Вихідні дані до роботи В рамках даної роботи робота ведеться з пристроєм Emotiv EPOC+._____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Аналітичний огляд, з висвітленням наступних питань: Робота з електроенцефалограми, потенціал Р300 і його стимуляція, організація робочої сесії, виявлення Р300 в електроенцефалограмі та інше. Основна частина, в якої висвітлити: Функціональні вимоги, не функціональні вимоги, опис даних, передоробка сигналу, сегментація сигналу, етап даталогічного проектування. Висновки. Перелік використаних джерел. _____
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання__30 березеня 2020 року._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.20	
2	Укладання і погодження з керівником плану і етапів виконання роботи	06.04.20	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	13.04.20	
4	Аналіз шляхів виконання завдання. Вибір і погодження керівником оптимального шляху	20.04.20	
5	Укладання та тестування програмного продукту	27.04.20	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	04.05.20	
7	Здача готової пояснювальної записки на кафедру	12.05.20	
8	Укладання доповіді і презентації	30.05.20	

Студент _____ ___Є. В. Воронін_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ _____Іванов В.Г_____
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-16д Воронін Є. В.

Науковий керівник

Доцент, к.т.н. _____ Іванов В.Г
Оцінка наукового керівника: _____

Рецензент _____
ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:
_____

Голова ЕК

_____ Лифар В.О.
підпис

РЕФЕРАТ

Робота містить: 40 сторінок основного тексту, 26 сторінок додатків, 7 рисунків, 20 використаних джерел.

Метою випускної кваліфікаційної роботи є Виявити найбільш зручне та доступне середовище розробки програмних модулів функціоналу інтерфейсу управління комп’ютером.

В результаті виконаної роботи, було: здійснено пошук рішень, що дозволяють здійснювати отримання даних ЕЕГ з Emotiv EPOC + та сформульовано ряд функціональних і не функціональних вимог до системи.

Система реалізована відповідно всім вимогам технічного завдання.

ЗМІСТ

ВСТУП

Актуальність досліджень. Мозок людини - вкрай складна біологічна структура. Що складається в середньому з 100 мільярдів нейронів, він є головним органом центральної нервової системи. Мозок відповідає за більшість процесів, що відбуваються в організмі людини, а також виконує когнітивну функцію, займаючись сприйняттям інформації, її обробкою і прийняттям рішень.

Розуміння того, як працює мозок - це тема, яка представляє інтерес нейрофізіологів і психологів вже протягом багатьох століть. Найбільше застосування знаходять результати, досягнуті в області електроенцефалографі: зміни електричних потенціалів, викликані активністю нейронів мозку, здатні дати величезну кількість корисної інформації про процеси, що відбуваються в мозку, і можуть бути досліджені математичними засобами. Ці зміни можна зареєструвати за допомогою розміщення електродів на поверхні шкіри голови, а сформований шляхом реєстрації таких змін тимчасової ряд називається електроенцефалограмою, або скорочено ЕЕГ.

Найбільш часто яку видобувають із ЕЕГ інформація - інтенсивність хвиль певної частоти в спектрі сигналу, що отримується шляхом. Перетворювання сигналу в спектр частот - так званих ритмів головного мозку. Така інформація може бути використання для аналізу станів мозку (сон, спокій, активна робота і т.п.), визначення поточної випробовуваної емоції і інших завдань, що стосуються дослідження загального стану мозку. Для деяких прикладних задач найбільш прийнятним виявляється вивчення сигналу на наявність потенціалів, пов'язаних з подіями (event related potentials, ERP) - різких змін в ЕЕГ, що виникають у відповідь на будь-якої аудіовізуальний або фізіологічний стимул. Особливо корисним є потенціал P300 - реакція мозку через приблизно 300 мс на стимул, розпізнається їм як

релевантний (наприклад, на знаходження потрібних слів в тексті при побіжному його прочитанні).

ЕЕГ, які будь-який інший сигнал природного походження, дуже схильний спотворень, що виникають в результаті сторонніх факторів. У разі ЕЕГ такі перешкоди, наприклад, викликаються електричними імпульсами, що посилаються м'язам, а також самим рухом м'язів, якщо такі знаходяться безпосередньо поблизу електродів. Дуже важливу роль відіграє правильне позиціонування електродів і їх хороший контакт з поверхнею голови. Подальшого поліпшення якості можна домогтися як апаратними засобами, так і за допомогою методів цифрової обробки сигналів.

З моменту становлення нейрофізіології як області та до недавнього часу дослідження в даній області були доступні лише університетам і мед. установам через дорожнечу використовуваного обладнання дня. У теперішній же час у зв'язку з появою комерційно доступних і портативних пристроїв - так званих нейрокомп’ютерних інтерфейсів (нейроінтерфейси, МКІ), здатних знімати ЕЕГ в домашніх умовах, нейрофізіологічні дослідження стали доступні ширшому колу ентузіастів. Такі нейроінтерфейси зазвичай представляють собор з 5-15 електродів, закріплених на поверхні шкіри голови і реєструють коливання напруги. Портативність таких пристроїв дозволяє їх використовувати в більш широкому ряді контекстів в порівнянні з медичним обладнанням - наприклад, при дослідженні активності мозку під час повсякденної діяльності.

Зростання таких технологій, як віртуальна (VR) і доповнена (AR) реальності, а також поширеність ситуацій, що вимагають використання hands-free пристроїв: водіння, робота з людьми з обмеженими можливостями - привели до збільшеного інтересу до нейрофізіології і нейроінтерфейси. Так, зараз існує досить велика кількість робіт, так чи інакше стосуються

реєстрації викликаних потенціалів в ЕЕГ - в тому числі і з застосуванням сучасних методів машинного навчання, однак застосування нейроінтерфейси для управління комп’ютером поки обмежується лише поруч синтетичних або досить вузьких завдань на зразок управління конкретними додатками [1; 2].

З огляду на зросту поширеність нейроінтерфейси і широку застосовність методів, призначених для розпізнавання ERP в сигналі, відсутність відкритого прикладного рішення для управління комп'ютером, адаптованого під загальні потреби і зручність кінцевого користувача, здається деяким недоглядом. Реалізація такого рішення буде корисна хоча б з тієї точки зору, що для того, щоб використовувати будь-яке програмне забезпечення з нейроінтерфейси, його не потрібно буде явно для цього адаптувати. При належному позиціонуванні така система здатна зробити нейроінтерфейси ще більш практично корисною технологією.

Об’єкт досліджень: В рамках даної роботи робота ведеться з пристроєм Emotiv EPOC +.

Предмет досліджень: МКІ був обраний з міркувань того, що він є одним з найбільш популярних комерційно доступних МКІ в даний час, а також володіє двома референтними датчиками і апаратним шумопоглинання.

Мета дослідження: Виявити найбільш зручне та доступне середовище розробки програмних модулів функціоналу інтерфейсу управління комп’ютером.

Завдання дослідження:

- Провести аналіз вимог до вирішення.
- Реалізувати алгоритм, що дозволяє витягувати з ЕЕГ необхідну для роботи з рішенням інформацію.
- Спроектувати і реалізувати прототип рішення, здатний виконувати просте взаємодія з операційною системою в реальному часу з використанням МКІ.
- Провести апробацію рішення.

Методологічна та теоретична основа дослідження: Розробка програмних модулів функціоналу інтерфейсу управління комп’ютером.

Практичне значення отриманих результатів. Розроблений прототип рішення, що дозволяє здійснювати введення тексту і позиціонування курсору на комп'ютері під керуванням ОС Windows.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Робота з електроенцефалограми

Мозок людини неоднорідний за своєю структурою - різні частки мозку відповідають за різні функції. Саме з цієї причини для зняття ЕЕГ використовується кілька датчиків, розташованих в місцях, що відповідають різним областям мозку.

Основними поняттями, на які спирається характеристика ЕЕГ, є: середня частота коливань, їх максимальна амплітуда і фаза, також оцінюються відмінності кривих ЕЕГ на різних каналах і їхня часова динаміка. Сумарна фонова електрограма кори і підкіркових утворень мозку пацієнта, варіюючись у залежності від рівня філогенетичного розвитку і відображаючи цитоархітектонічні і функціональні особливості структур мозку, також складається з різних за частотою повільних коливань.

У деяких частинах можна виявити періодичні низькочастотні коливання - так звані ритми головного мозку. Інтенсивність таких коливань безпосередньо залежить від того, чим зайнятий мозок у поточний момент. В роботі [3] Наведено короткий огляд характеристик частотного спектра ЕЕГ людини і обговорюється їх зв'язок з різними станами людського тіла, такими як сон, активна діяльність, концентрація, стрес і іншими. В роботі [4] Наведено приклад використання методів частотної (дозволяє виділити конкретні ритми) і просторової (дозволяє локалізувати сигнал, специфічний для будь-якої з часткою) фільтрації для побудови вектора ознак в завданні класифікації поточного психічного стану.

Дослідження спектра сигналу - не єдиний спосіб отримати корисної інформацію з ЕЕГ. Найчастіше більш корисним виявляється дослідження сигналу на наявність потенціалів, пов'язаних з подіями - реакцій мозку на

який-небудь зовнішній стимул. Докладний огляд найбільш популярних з таких потенціалів наведено в роботі [5].

1.2 Потенціал P300 і його стимуляція

Для нашої задачі найбільш корисним виявляється потенціал P300, який отримав свою назву завдяки тому, що виникає через 300 мілісекунд після рідкісного і релевантного стимулу (наприклад, при візуальному виявленні потрібного об'єкту серед безлічі не релевантних). Вибір P300 обумовлений двома причинами:

• По-перше, завдяки простоті його стимуляції досить легко заснувати сесію запису даних таким чином, щоб у відповідь на деякий стимул P300 гарантовано вироблявся, а при відсутності - не виробляється [1]. Це забезпечує його універсальність, роблячи придатним для використання в широкому ряді завдань, які можна, можливо звести до задачі вибору потрібного класу з декількох;

деякий стимул P300 гарантовано вироблявся, а при відсутності - не виробляється [1]. Це забезпечує його універсальність, роблячи придатним для використання в широкому ряді завдань, які можна, можливо звести до задачі вибору потрібного класу з декількох;

• По-друге, завдяки своїй широкій застосовності існує досить багато робіт, заснованих на виявленні P300 в сигналі. Конкретні підходи різняться в залежності від використовуваного обладнання, поставлених задач і стану випробуваного [6], Але ідеї виявляються застосовні і в інших роботах.

На графіку 1 (рис.1) наведено усереднений вид потенціалу P300 і проведено порівняння зі звичайним сигналом без будь-яких попередніх зонних стимулів.

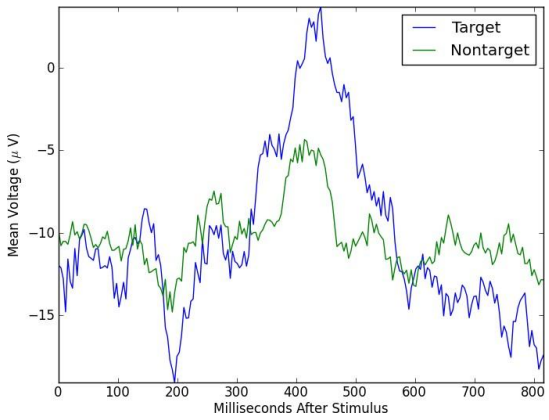

Рис. 1.1: Вид потенціалу P300 [7]

стимуляція P300 зазвичай здійснюється за допомогою, так званої oddball-парадигми [1] - випробуваному послідовно у випадковому порядку показуються релевантні і не релевантні стимули, причому релевантні стимули показуються рідше. Від випробуваного потрібно подумки відзначати поява релевантного стимулу (наприклад, підрахувати їх), що і стимулює виникнення даного потенціалу. Oddball-парадигма легко узагальнюється на випадок мультікласової класифікації, коли від випробуваного потрібно здійснити вибір між кількома рівнозначними класами.

Практично всі рішення, що базуються на розпізнаванні P300 у сигналі, складаються з чотирьох етапів.

1. Первинна добірка сигналу (згладжування, фільтрація).

2. Сегментація сигналу з урахуванням інформації про стимули.

3. Опціонально - побудова вектора ознак (наприклад: усереднення вимірювань декількох стимулів, результати застосування банку частотних фільтрів, коефіцієнти wavelet-перетворення [8], Докоефіцієнти авторегресії [9] Та ін.).

4. Навчання і використання бінарного класифікатора (SVM, LDA, нейронні мережі та ін.) для визначення наявності P300 в сегменті.

1.3 Організація робочої сесії

Серед параметрів робочої сесії, безпосередньо впливають на якість розпізнавання і час, що витрачається на одну ітерацію розпізнавання, можна виділити наступні:

• Кількість повторень релевантного стимулу - якщо воно велике, це підвищує ймовірність коректного розпізнавання стимулу, але вимагає більшої кількості часу на проведення однієї ітерації розпізнавання.

• Відстань між стимулами (target-to-target interval, TTI)- чим воно менше, тим менш виражені будь-які виникають потенціали, але скорочується час, що витрачається на розпізнавання.

• Кількість класів - якщо класів мало, то стимул, який відповідає цьому класу, виникає частіше, і може виявитися менш виражений.

Це - не єдині фактори, що впливають на якість процесу розпізнавання P300, але вони є найбільш важливі, тому що вони, по-перше, не залежать від психічного стану випробуваного, а по-друге можуть бути при необхідності змінені. Більш детальний огляд факторів, що впливають на якість даних ЕЕГ, наведений у статті [6].

1.4 Виявлення P300 в електроенцефалограмі

У роботах [2;10] Наводиться детальний огляд алгоритмів, застосовуваних на етапі класифікації. Обговорюється можливість застосування класичних методів машинного навчання (LDA, SVM) до задачі класифікації сегмента за ознакою наявності P300, обговорюються способи

побудови вектора ознак за допомогою як простих засобів обробки сигналів, так і спеціалізованих алгоритмів (аналіз головних компонент, генетичні алгоритми, адаптивна фільтрація та ін.) Стаття[11] Пропонує використовувати так званий shape feature vector - вектор ознак на основі геометричної інформації сегмента ЕЕГ.

1.5 Поліпшення якості розпізнавання P300

В роботі [12] Обговорюється застосування методів цифрової обробки сигналів (нормалізація щодо середнього, фільтрація частот та ін.) Для поліпшення якості одержуваного сигналу і усунення проблем на кшталт «пливе середнього» (зміщення значення середнього арифметичного сигналу на тимчасовому вікні щодо первісного значення з течією часу).

В роботі [13] Обговорюється, як виглядають артефакти ЕЕГ, породжені серцебиттям, скороченням м'язів і рухом очей. Для знаходження аномалій пропонується використовувати метод головних компонент (PCA) і метод незалежних компонент (ICA) з подальшою фільтрацією компонент з аномально високою амплітудою.

Стаття [14] Демонструє, як для поліпшення якості розпізнавання P300 застосовується нормалізація сигналу, його згладжування за допомогою фільтра і придушення немаксімумов (non-maximum suppression) для позову півсигналу, які є потенційними кандидатами на те, щоб бути визначені, як P300.

1.6 Існуючі рішення

EPOC має 14 електродів (в порівнянні з 19 електродами стандартного медичного ЕЕГ і 3 в OCZ NIA). Самі електроди є пасивними - вловлюють сигнал і передають його далі, кріпляться на поверхні шкіри (непогружной

інтерфейс) і вимагають змочування спеціальною рідиною для кращого контакту (мокрий інтерфейс). Також має двовісний гіроскоп для вимірювання обертання голови.

Гарнітуру спочатку потрібно «навчити» розпізнавати яка думка повинна відповідати певним дії. Прилад може вимірювати чотири види даних, але деякі користувачі говорять, що головним чином знімаються дані з виразу обличчя:

Розуміння думки (Cognitiv Suite): уявляється 12 видів руху - 6 напрямів (вліво, вправо, вгору, вниз, вперед і «зум») і 6 поворотів (обертання за і проти годинникової стрілки, поворот наліво і направо, нахил вперед і назад) - плюс ще одна візуалізація «зникнення», яку виявляють в Мю-ритмі. Ідеомоторні реакції або більш сильні сторонні струми ЕЕГ - ці «уявні» команди фактично стають «гарячими клавішами». Відео «The Game» від співробітників Emotiv показують високий ступінь труднощі в адаптації і правильне мислення навіть у досвідчених користувачів. Через складні алгоритмів виявлення викликів, є невелике відставання у виявленні думки.

Емоції (Affectiv Suite): «Порушення», «Захоплення / Нудьга», «Замисленість», і «Розчарування» зараз можна виміряти. Emotiv визнає, що ці назви можуть відображати не саме ті емоції, які використовуються, і кажуть, що можуть уточнити назви, перед виходом на ринок.

Вираз обличчя (Expressiv Suite): Індивідуальні позиції повік і брів, положення очей в горизонтальній площині, посмішки, сміх, зціпити зубів і посмішки. Інші вирази можуть бути додані до випуску. Вирази виявляються датчиками ЕЕГ збирають сигнали м'язів обличчя, а не шляхом читання мозкових хвиль. На відміну від зчитування психічної активності, виявити зміни таким чином можна дуже швидко (10 мс) надаючи вирішальну перевагу і роблячи їх придатними для швидких темпів гри в жанрі FPS.

Обертання головою: кутову швидкість голови можна виміряти за допомогою нишпорення і тангажу (але не крену). Це реєструється гіроскопами, і не пов'язано з особливостями ЕЕГ.

В роботі [1] Надано огляд найбільш популярних рішень з області людино-комп'ютерного взаємодії, основною ідеєю якої є реєстрація стимульованого штучно потенціалу P300. Велика увага присвячена питанню набору тексту: досліджуються способи конструювання інтерфейсу таким чином, щоб максимізувати точність вибору потрібної літери і мінімізувати час, витрачений на то, щоб зробити вибір.

У статті [15] Представлений простий приклад використання P300 і класифікатора на основі методу опорних векторів (SVM) для реалізації простого детектора брехні.

Існує публічний набір даних [16], Отриманий з використанням медичного обладнання та використаний для завдання побудови системи введення тексту на базі потенціалу P300. Є ряд рішень, присвячених введенню тексту за допомогою P300 і використовують даний набір даних для навчання і тестування відповідних класифікаторів - [17] (На базі ансамблю SVM), [18] (На базі градієнтного бустінга).

Компанія Emotiv надає два програмних рішення, дозволяючи здійснювати роботу з ЕЕГ.

• EmotivPRO[19] - пропріетарна інтегроване середовище для роботи з даними ЕЕГ. Дозволяє здійснювати запис, перегляд і сегментацію і розмітку сигналу, а також здійснювати аналіз ритмів головного мозку. Має можливість експортування даних в форматі CSV.

• Emotiv Community SDK[20] - бібліотека, що дозволяє використовувати можливості, аналогічні таким в EmotivPRO, у власних додатках. Вимагає наявності ліцензії на EmotivPRO для роботи з «сирими»

даними ЕЕГ. Підтримка припинена з кінця 2018 року, наявними користувачам пропонується користуватися новим хмарним сервісом Emotiv Cortex.

1.7 Рішення з відкритим вихідним кодом

В силу ліцензійних обмежень, що накладаються Emotiv на використанніня власних рішень для роботи з Emotiv EPOC + в інших додатках і складності розробки альтернативного рішення «з нуля», потрібно було провести пошук альтернативних рішень з відкритим вихідним кодом і більш вільною ліцензією.

В результаті пошуку наявних рішень у відкритих репозиторіях (GitHub, GitLab, BitBucket) були знайдені такі бібліотеки із заявленою підтримкою Emotiv EPOC +.

• EmoKit(https://github.com/openyou/emokit, 418 stars) - одне з перших рішень. Володіє рядом мінусів: не оновлюється з 2017 року, ні підтримки пристроїв Emotiv EPOC +, випущених пізніше 2017 року.

• CyKit(https://github.com/CymatiCorp/CyKit, 38 stars) - ідейний послідовник EmoKit. Підтримує роботу з усіма пристроями Emotiv, існуючими на сьогоднішній день. Унаслідок залежності від pywinusb дана бібліотека працює тільки на ОС Windows, проте це компенсується тим, що існує можливість використовувати бібліотеку як сервер, здатний здійснювати передачу даних в реальному часі за допомогою протоколу WebSocket.

В кінцевому підсумку для зняття даних була обрана бібліотека CyKit - як єдине знайдене рішення, здатне працювати з сучасними моделями Emotiv EPOC +.

1.8 OpenViBE

Також варто згадати про OpenViBE [21] - інструментарії для роботи з ЕЕГ, розробленим французьким Національним інститутом досліджень в інформатиці та автоматики (INRIA). OpenViBE - це програмна платформа, призначена для проектування, тестування та використання інтерфейсів мозок-комп'ютер. Пакет включає інструмент «Дизайнер» для створення та запуску користувацьких додатків, а також кілька попередньо налаштованих та демонстраційних програм, готових до використання. OpenViBE призначений для роботи з даними ЕЕГ в реальному часі і представляє набір інструментів для преобробки сигналу, його візуалізації та подальшої роботи з ним. Для розробки сценаріїв OpenViBE надає спеціальний графічний мову, що робить його вкрай зручним для прототипування. Один з компонентів OpenViBE - Acquisition Server – надає можливість отримання даних ЕЕГ з різних джерел: від конкретних моделей нейроінтерфейси до прийому довільних числових даних через Telnet. Отримання даних від Emotiv EPOC + не підтримує OpenViBE безпосередньо, але завдяки підтримці WebSocket є можливість використовувати OpenViBE спільно з CyKit.

РОЗДІЛ 2. ЗАСОБ РОЗРОБКИ ТА ВИМОГИ

В даному розділі описуються вимоги, висунуті до прототипу. Вони розділені на дві групи - функціональні і не функціональні.

Функціональні вимоги визначають мінімальну функціональність, яку повинен реалізовувати прототип, щоб він ніс практичну значимість.

Не функціональні вимоги покликані обмежити кількість побічних технічних завдань, пов'язаних з перенесенням і універсальністю рішення, і дозволяють сфокусуватися на основних завданнях, поставлених для досягнення мети цієї роботи.

2.1 Функціональні вимоги

Взаємодія з ОС. Рішення повинно дозволяти здійснювати два види взаємодії з ОС: введення тексту і позиціонування курсору. Взаємодія має бути організовано у вигляді циклу послідовних дій призначеного для користувача введення.

Будь-які допоміжні елементи призначеного для користувача інтерфейсу, що допомагають здійснювати взаємодію (наприклад, здійснює генерацію візуальних стимулів), не повинні заважати сприйняттю поточного вмісту на екрані і не повинні бути доступний для взаємодії безпосередньо.

Обробка сигналу в режимі реального часу. В силу того, що будь-які зовнішні стимули (в тому числі і візуальні) негайно знаходять своє відображення в ЕЕГ, робота з сигналом повинна бути організована в режимі реального часу.

Це означає, що в будь-який конкретний момент ми повинні мати можливість отримати доступ до актуальних вимірам. Так як потенціали,

пов'язані з подіями, виникають через проміжок від декількох десятків від декількох сотень мілісекунд, затримка в 50 мс вже буде представляти проблеми для їх коректного розпізнавання.

2.2 Не функціональні вимоги

Сигналізація про помилки. При виникненні будь-яких проблем з обладнанням, а також інших проблем, що перешкоджають подальшій роботі, слід негайно сповістити про це користувача і призупинити сесію доти, поки проблема не буде вирішена.

Точність. В силу того, що ЕЕГ - сигнал природного походження, його використання для управління комп'ютером пов'язане з певними труднощами. Для того, щоб рішення було практично застосовним, потрібно мінімізувати кількість випадків, коли система неправильно розпізнає, яка з дій було вибрано користувачем.

Простота використання. Рішення повинно вимагати від користувача мінімальних зусиль з розгортання і запуску. Єдине, що повинно вимагатися від користувача для забезпечення коректного функціонування рішення - вказати адресу сервера CyKit і записати дані тренувальної сесії.

Цільова платформа. В якості цільової платформи для розробки обрана OS Windows через те, що вона є найбільш популярною серед ОС споживчого сегмента [22].

Вибір конкретної платформи дозволить скоротити час, що витрачається на реалізацію трансляції дій користувача в конкретні системні виклики і дасть можливість сфокусуватися на більш завданнях. У порівнянні з іншими ОС, вибір Windows як цільової платформи дозволить збільшити охоплення потенційної користувальницької бази рішення.

Використання потенціалу P300. В основі алгоритму, що витягує з ЕЕГ інформацію, необхідну для здійснення призначеного для користувача введення, має лежати виявлення потенціалу P300.

2.3 Опис даних

Нейроінтерфейс Emotiv EPOC + володіє 16 датчиками, з яких два (P3 і P4) є референтними і призначені для корекції сигналу. Розташування датчиків приведено на рисунку 2.

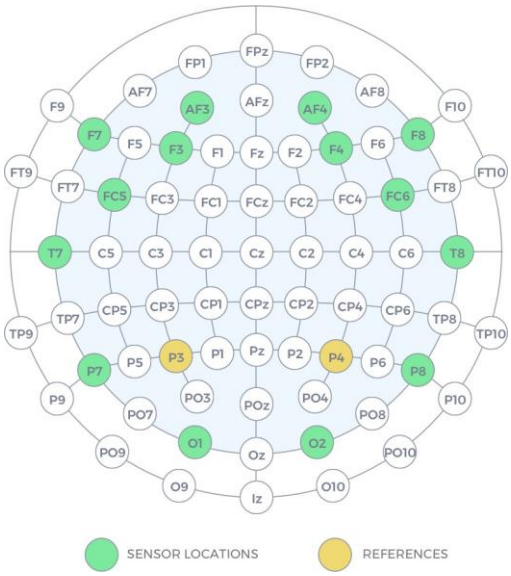

Рис. 2.1 - Розташування датчиків Emotiv EPOC + (з керівництва [23])

Дані з електродів знімаються з внутрішньої частотою дискретизації 2048 Гц. Нейроінтерфейси проводиться апаратне видалення перешкод від мереж змінного струму за допомогою двох фільтрів з частотами 50 Гц і 60 Гц відповідно. Нормалізація сигналу щодо значення референтних датчиків також виконується апаратно. Після апаратної фільтрації проводиться зниження частоти дискретизації до заданої частоти (128 Гц або 256 Гц в залежності від обраного режиму роботи пристрою). На кодування одного

каналу даних виділяється 16 біт, найменшому значущому біту відповідає 0.13 мікровольт.

Результуючі дані з усіх 14 каналів доповнюються лічильником поточного вимірювання, обнуляє кожні 128 (або 256, в залежності від обраного режиму) вимірювань.

2.4 Передоробка сигналу

Робота над даними, знятими з нейроінтерфейси, вимагає попередньо котельної обробки. Для цього були виконані наступні кроки:

• Застосування до сигналу смугового фільтра з діапазоном частот 0.4-30Гц;

• підвибірки кожного другого виміру (64 Гц);

• видалення артефактів за допомогою ICA.

Смуговий фільтр застосовується в зв'язку з тим, що P300 - низькочастотний сплеск, а видалення частот менше 0.4 Гц сприяє виправленню проблеми baseline drift (пливе середнє).

Підвибірки береться з метою видалення надлишкової інформації: по теорема Найквиста-Шеннона сигнал з частотою дискретизації 64 Гц може нести в собі інформацію про частоти до 32 Гц. Так як до цього був застосований смуговий фільтр і відсічені всі частоти вище 30 Гц, можна закодувати отримані дані з частотою дискретизації 64 Гц без особливих втрат якості.

Так як артефакти зазвичай не корелюють з сигналом, за допомогою ICA поділяємо сигнал на незалежні компоненти - артефакти і смислове частина. Артефакти мають велику амплітуду, ніж сигнал, тому відсортуємо компоненти по максимальній амплітуді і відкинемо кілька перших компонент. Однак є ризик випадково відсікти потенціал P300

- прибирати дуже багато компонент можна.

2.5 Сегментація сигналу

Процес сегментації виглядає наступним чином:

• з ЕЕГ вирізається одна секунда даних після отриманого стимулу;

• відгуки на стимули одного класу об'єднуються в один вектор ознак шляхом усереднення сигналу (щодо кожного каналу);

• дані всіх каналів об'єднуються в один вектор ознак;

• класифікатором передаються вектора ознак без будь-яких допопередачі змін.

Такі параметри сесії, як TTI і кількість повторень сегмента можуть бути налаштовані перед проведенням експерименту.

2.6 Випробувані класифікатори

Для вирішення завдання класифікації сигналу були випробувані популярні класифікатори, що зустрічаються в рішенні подібних задач -SVM, LDA і повно зв’язні нейронні мережі.

SVM (Support Vector Machine, метод опорних векторів) був обраний через те, що він добре працює на даних з високою розмірністю. Однак для того, щоб показати хорошу точність в завданні бінарної класифікації, часто доводиться підбирати параметри класифікатора (зокрема, тип ядра) або займатися завданням вилучення ознак з необробленого вектора.

Основна ідея методу - переклад вихідних векторів в простір більш високої розмірності і пошук розділяє гиперплоскости з максимальним зазором в цьому просторі. Дві паралельні гіперплощини будуються по обидва

боки гіперплощини, що розділяє класи. Розділяє гиперплоскостью буде гіперплоскость, максимізує відстань до двох паралельних гіперплоскостей. Алгоритм працює в припущенні, що чим більша різниця або відстань між цими паралельними гіперплоскостями, тим менше буде середня помилка класифікатора.

LDA (Linear Discriminant Analysis, лінійний дискримінантний аналіз) випробуваний в зв'язку з тим, що він з високою точністю вирішує завдання пошуку критерію, за яким два різних класи легко відрізняються одне від одного. В даному випадку це клас фрагментів сигналу, в яких присутня P300, і клас фрагментів сигналу, в яких P300 не зустрічається. Найчастіше такі класи можна відрізнити візуально.

ЛДА тісно пов'язаний також с методом головних компонент (МГК, англ. Principal Component Analysis, PCA) і факторний аналіз тим, що вони шукають лінійні комбінації змінних, які найкращим чином пояснюють дані. ЛДА явно намагається моделювати різницю між класами даних. МГК, з іншого боку, не бере до уваги будь-яку різницю в класах, а факторний аналіз будує комбінації ознак, спираючи скоріше на відмінності, а не на схожості. Дискримінантний аналіз відрізняється також від факторного аналізу тим, що не є незалежною технікою - для його роботи має бути визначено різницю між незалежними змінними і залежними змінними (останні називаються також критеріальними змінними).

ЛДА працює, коли вимірювання, зроблені на незалежних змінних для кожного спостереження, є безперервними величинами. Коли маємо справу з якісними незалежними змінними, еквівалентній технікою є дискримінантний аналіз відповідностей.

Дискримінантний аналіз використовується, коли групи відомі апріорі (на

відміну від кластерного аналізу). Кожен випадок повинен мати значення в одній або декількох заходів кількісного передбачення і значення на груповий міру. Висловлюючись простими термінами, аналіз дискримінантних функцій є класифікацією, що розбиває об'єкти на групи, класи або категорії деякого типу.

Також були використані повно зв'язні нейронні мережі, так як вони добре справляються із завданням навчання на прихованих ознаках.

2.7 Реалізація прототипу системи

Незважаючи на те, що засобами описаних в главі 2.3 рішень можливо організувати запис даних ЕЕГ в файл для подальшої роботи з ними в автономному режимі, для проведення повноцінних експериментів з призначеним для користувача введенням в реальному часі перед реалізацією алгоритму розпізнавання P300 потрібно реалізувати частину рішення, відповідальну за отримання і обробку даних в реальному часі, а також призначений для користувача інтерфейс.

2.8 Загальний вигляд

Загальна архітектура рішення (у вигляді діаграми компонентів UML) представлена на малюнку3. На ній показано, що рішення складається з трьох основних модулів.

• модуль realtime відповідає за отримання даних, їх первинну предобработку і сегментацію, а також сповіщення для користувача інтерфейсу про необхідність генерації стимулу.

• модуль app відповідає за основну логіку програми (підключення до сервера CyKit, запуск модуля realtime, реалізація інтерфейсу).

• модуль model надає алгоритм розпізнавання P300.

Для отримання даних ЕЕГ з нейроінтерфейси використовується бібліотека CyKit. Вона ж надає можливість передачі даних через протокол TCP для подальшої обробки.

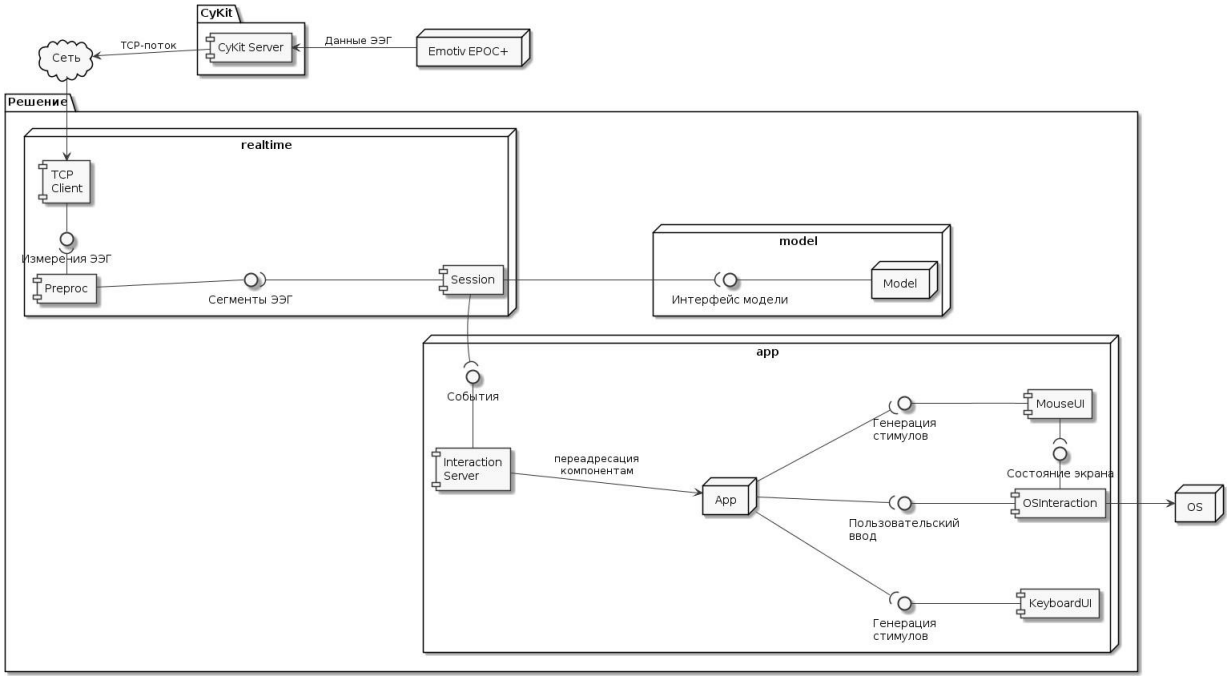

рис. 2.2 - діаграма компонентів рішення

Поділ логіки додатка на окремі частини обумовлено тим, що призначений для користувача введення і генерація візуальних стимулів є деякими викликаються подіями, що дозволяє нам використовувати для реалізації призначеного для користувача інтерфейсу і службової логіки подієвоорієнтований фреймворк (в нашому випадку Qt). Навпаки, обробка сигналу вимагає безперервного взаємодії з ним.

Ізоляція обробки сигналу дозволяє уникнути ряду проблем: якби все додаток цілком було реалізовано у вигляді робочого циклу, в реальному часі обробляє дані ЕЕГ, це викликало б труднощі при розробці користувальницького інтерфейсу. Навпаки, якби ми використовували подієвоорієнтований фреймворк і реалізували отримання вимірювань як

окремі події, то це б, по-перше, викликало б значне навантаження на чергу подій (як мінімум 128 подій в секунду при частоті дискретизації пристрою 128 Гц), а у друге - збільшило б затримку між реєстрацією вимірювання і його обробкою.

Розглянемо модулі app і realtime більш докладно.

2.9 Модуль realtime

Основним класом модуля realtime є клас Session, що організує високорівневу логіку взаємодії з комп'ютером:

• отримання сегментів ЕЕГ - шляхом посилки асинхронного виклику

get_segment класу Preproc і очікування результату на вимогу;

• взаємодії з моделлю;

• оповіщення додатки про необхідність генерації візуального стимуляції шляхом посилки батьківського процесу повідомлення;

• оповіщення додатки про обраний дії - шляхом посилки батьківському процесу повідомлення.

Клас Session також надає метод run, який є точкою входу дочірнього процесу, породжуваного додатком при запуску сесії. Клас Preproc инкапсулирует логіку первинної предобработки даних і сегментації сигналу. Він надає інтерфейс на базі асинхронних викликів, що дозволяє обробляти одержувані від пристрою вимірювання, виконувати їх предобработку і доповнювати запити на сегменти новими даними. Повертаються класом сегменти представляють собою awaitable значення, які можуть бути використані, як тільки отримано необхідну кількість даних.

Клас TCPClient займається отриманням і обробкою TCP-потоку, поЛучано від CyKit. Він виконується в окремому потоці і реалізує патерн «спостерігач», реалізуючи функціональність оповіщення підписаних на нього класів про те, що доступні нові дані.

Типовий сценарій отримання сегмента показаний на малюнку4. На діаграмі можна побачити, як об'єкт класу Session посилає TCP-клієнту повідомлення про необхідність встановлення з'єднання з CyKit. Далі Session підписує Preproc на оновлення сервера - TCP-клієнт при отриманні чергового сегмента сповіщає про це об'єкт класу Preproc. Безпосередньо отримання сегмента від Preproc проводиться за допомогою виклику асинхронного методу get_segment, результат якого повертається в той момент, коли зібрана необхідна кількість даних.

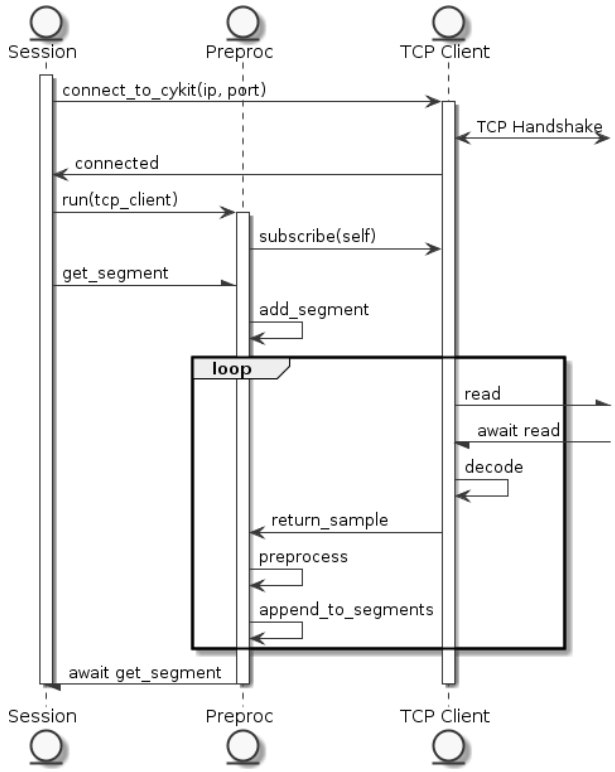

рис. 2.3 - Сценарій отримання сегмента

2.10 Модуль app

Вся інша логіка програми, яка не потребує взаємодії з данними в реальному часі, поміщена в модуль App:

• службовий призначений для користувача інтерфейс (запустити / зупинити край, то налаштувати установки сервера CyKit);

• елементи інтерфейсу, що відповідають за відображення візуальних стимулів (Класи MouseUI, KeyboardUI);

• клас, інкапсулює логіку взаємодії з ОС (OSInteraction).

Реалізація взаємодії між процесами здійснена за допомогою протоколу TCP. Для отримання повідомлень від дочірнього процесу реалізований клас InteractionServer, здатний приймати від дочірнього процесу повідомлення про необхідність генерації стимулу і про обраний дії.

2.11 Користувальницький інтерфейс

Додаток здатний працювати в двох режимах.

• Робочий - алгоритм розпізнавання P300 знаходить, які стимули являють релевантними, і в залежності від цього транслює дії користувача в системні виклики.

• Тренувальний - від користувача потрібно вибрати заздалегідь зазначений клас. Відповідні дані ЕЕГ збираються і використовуються для навчання класифікатора.

Для контролю поточного обраного режиму, а також налаштування параметрів підключення до сервера CyKit використовується меню програми, доступне через значок в системному.

В рамках даного рішення для введення тексту використовується наступний підхід.

Рис. 2.4 - Інтерфейс для введення тексту

• Символи, доступні для введення, організовані в матрицю 6x6 (рис.2.4).

• В якості стимулів виступає підсвічування шпальти чи рядки (всього 12 класів). Від користувача потрібно сфокусуватися на потрібному символі і подумки відзначати, коли підсвічуються містять його стовпець або рядок.

• Шляхом застосування алгоритму розпізнавання P300 знаходяться два релевантних класу, відповідні релевантним колонку і рядку.

Спочатку був розроблений варіант дозволяє здійснювати переміщення курсору на фіксовану кількість пікселів в одному з чотирьох напрямків, а також здійснювати емуляцію клацання лівою або правою кнопкою миші (рис. 2.5)

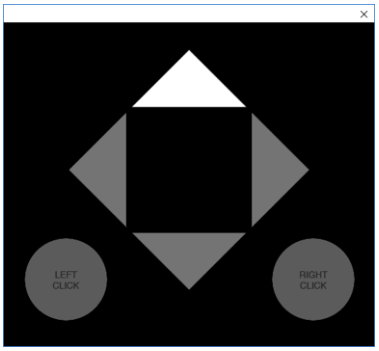

Рис. 2.5 - UI для пересування курсору, старий варіант

Однак, такий варіант мав очевидні недоліки - на переміщення курсору між віддаленими частинами екрана доводилося затрачати дуже багато дій.

Більш зручним виявився наступний підхід, коли простір ділиться на чверті і користувачеві пропонується вибрати між ними. Після вибору потрібної частини екрану процес повторюється рекурсивно до тих пір, поки курсор не позиціонуватиметься точно. Даний підхід дозволяє вимагає меншу кількість дій: наприклад, на дисплеях з роздільною здатністю FullHD потрібно здійснити всього 9 ітерацій. Для того, щоб гарантувати стимуляцію P300, замість того, щоб підсвічувати потрібну частину робочого столу безпосередньо, активна чверть відображається в збільшеному вигляді в окремому вікні (рис. 2.6).

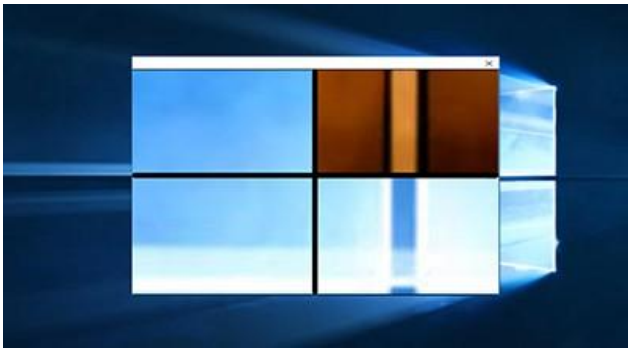

Рис. 2.6 - Інтерфейс для пересування курсору, новий варіант

Нейроінтерфейс (або інтерфейс «мозок - комп'ютер») - так називається пристрій для обміну інформацією між мозком і зовнішнім пристроєм. Як об'єкт управління може виступати не тільки комп'ютер, але і будь-яке інше електронний пристрій: квадрокоптер, система «розумного будинку», промисловий робот або бойової дрон, екзоскелет і навіть штучні органи чуття.

Медицина на даний момент є основною сферою застосування нейроінтерфейси. Тут інтерфейс «мозок - комп'ютер» відкриває нові можливості в області протезування та реабілітації інвалідів з різними моторними порушеннями. Наприклад, після інсульту багато пацієнтів не можуть говорити. У цій ситуації нейроинтерфейс виступає розумним посередником між мозком і зовнішньої реальністю, єдиним засобом спілкування.

Паралізовані пацієнти за допомогою такого пристрою можуть управляти протезом і інвалідним візком або навіть механічним екзоскелетом. Мабуть, найкраще наочний доказ фантастичних можливостей цієї технології сталося в 2014 році. Тоді Чемпіонат світу з футболу в Бразилії відкрив ударом по м'ячу Джуліано Пінто - людина з паралічем нижніх кінцівок. Зробив він це за допомогою екзоскелета, керованого силою думки.

Нейроінтерфейси впевнено входять в повсякденне життя і розширюють області використання. Сьогодні до технології «мозок - комп'ютер» починає проявляти інтерес не тільки медицина, а й розважальна галузь з її комп'ютерними «іграшками», промислове виробництво, пристрої «розумного будинку», роботехника.

Згідно з дослідженням Allied Market Research, ринок інтерфейсів «мозок - комп'ютер» зростає випереджаючими темпами і вже в 2020 році складе близько 1,46 млрд доларів.

Нейроінтерфейси впевнено входять в повсякденне життя і розширюють області використання. Сьогодні до технології «мозок - комп'ютер» починає проявляти інтерес не тільки медицина, а й розважальна галузь з її комп'ютерними «іграшками», промислове виробництво, пристрої «розумного будинку», роботехника.

Згідно з дослідженням Allied Market Research, ринок інтерфейсів «мозок - комп'ютер» зростає випереджаючими темпами і вже в 2020 році складе близько 1,46 млрд доларів.

Можна сказати, що історія інтерфейсу «мозок - комп'ютер» налічує понад сто років. Ще в 1875 році, задовго до винаходу самого комп'ютера, англійський фізіолог і хірург Річард Кетон виявив електричні сигнали на поверхні мозку тварини. У 50-ті роки минулого століття з'явився перший нейроинтерфейс. Їм прийнято вважати Stimoceiver - електродні пристрій, який управлялося по бездротовій мережі за допомогою FM-радіо. Воно було винайдено іспанським та американським вченим Хосе Дельгадо і випробувано в мозку бика. Демонстрація можливостей нового пристрою була дуже ефектною - на арені для кориди. Дельгадо вийшов проти бика, а коли той побіг на нього, натиснув кнопку на пульті управління - вперше вдалося змінити напрямок руху тварини за допомогою нейроінтерфейси.

У 1998 році був впроваджений перший нейроинтерфейс в мозок людини. Пацієнтом став американський художник і музикант Джонні Рей. Думаючи або представляючи руху рук, Рей керував курсором на екрані комп'ютера.

Але справжній прорив стався кілька років тому, коли з'явилися досить потужні комп'ютери і нові алгоритми. Якщо раніше можна було розшифровувати тільки найпростіші наміри, наприклад, хоче людина поворухнути правою рукою або лівою, то сучасний нейроинтерфейс може керувати навіть окремими пальцями протеза руки. Для цього потрібно

впровадити на ділянці мозку, що відповідає за рух рук, більше 100 електродів.

Звичайно, нові технології надали нові неймовірні можливості в цій сфері, але принципова ідея нейроінтерфейси така ж, як і півстоліття тому. В інтерфейсі «мозок - комп'ютер» немає нічого містичного: технологія дозволяє реєструвати електричну активність мозку і перетворювати її в команди для зовнішніх пристроїв.

«Це не телепатія і не телекінез: в нейроінтерфейси уявні команди людини розшифровуються за записом електричної активності його мозку, або електроенцефалограми. Тієї самої, яку записують у кожній поліклініці », - пояснює психофізіолог Олександр Каплан, завідувач лабораторії нейрофізіології і нейроінтерфейси біологічного факультету МГУ.

Зчитування сигналів мозку виробляється за допомогою інвазивних (імплантуються в мозок пацієнта) датчиків або неінвазивних датчиків, які реєструють ЕЕГ з поверхні голови.

Отже, для інвазивного нейроінтерфейси потрібна операція: електроди імплантуються прямо в кору мозку. Виглядають вони як маленька платівка, приблизно п'ять на п'ять міліметрів, яка покрита сотнями голочок-електродів. Вони реєструють електричну активність окремих нервових клітин в тому місці, куди впроваджені. Такі датчики відрізняються більш сильним сигналом, однак інвазійних втручання пов'язане з наслідками для здоров'я людини. Навіть відмінні характеристики датчиків нового покоління можуть викликати ряд проблем: ризик запалень, необхідність повторної імплантації через відмирання нейронів і навіть такі незрозумілі наслідки, як епілепсія. Тому такі інтерфейси використовують в крайніх випадках, для тяжкохворих пацієнтів, яким не можуть допомогти інші методи.

Неінвазивний нейроинтерфейс не припускав вторгнення в організм - електроди прикріплюють до шкіри голови. Незважаючи на те що мозок розташовується глибоко в черепі, електричні поля, створювані нервовими клітинами, уловлюються електродами на поверхні голови. Цей метод вже давно застосовується при знятті електроенцефалографії. З використанням нейрогарнітури можливо побудувати інтерфейс «мозок - комп'ютер», що забезпечує точність розпізнавання команд користувача до 95%.

У свою чергу, неінвазивні нейроінтерфейси можуть бути на «мокрих» і «сухих» електродах. У першому випадку електроди з подушечками потрібно змочувати і лише потім прикріплювати до голови. Як відомо, рідина служить провідником електрики і полегшує зняття даних. Однак у такого методу є недоліки, і це не тільки мокре волосся.

Нейроінтерфейси на сухих електродах виглядають у вигляді шолома, який можна легко надіти без будь-якої додаткової допомоги та підготовки. Спеціальні електроди не вимагають використання електропровідного гелю, при цьому високу якість реєстрованого сигналу забезпечує система активного придушення перешкод. Наприклад, подібний нейроинтерфейс розробив концерн «Автоматика» Держкорпорації Ростех.

РОЗДІЛ 3 ПРОЕКТНА ЧАСТИНА

МКІ вчені плідно впроваджують в медицину, зокрема - в реабілітацію людей з ушкодженнями спинного мозку і втратили здатність самостійно пересуватися. Останні винаходи швейцарських вчених успішно пройшли тестування на тваринах. Замість того, щоб виправити пошкоджену ділянку спинного мозку, вчені запропонували революційну ідею - обійти розрив стороною, створивши штучний біоелектронних міст. Крім цього передбачається широке впровадження електромеханічна протезів, якими люди зможуть керувати за функціональністю здоровій кінцівці. Це можливо завдяки впровадженню в мозок тварини або людини кількох десятків електродів, які передають нервові імпульси високотехнологічному протезу.

Зовнішні датчики МКІ спрямовані на те, щоб зчитувати хвильові функції мозку, що не впроваджуючи електроди в тіло людини. Основних джерел отримання подібної інформації три: електроенціфалографія, функціональна магнітно-резонансна інтроскопія і оптична друкарня.

У 2019 заснований підприємцем Ілона Маском стартап Neuralink повідомив про готовність у 2020 році почати тестування на людях імплантатів, які дозволять знерухомлених пацієнтам управляти комп'ютером і смартфоном за допомогою власних думок. У липні 2019 року компанія провела презентацію нової технології, що дозволяє реєструвати активність мозку щура за допомогою тисяч імплантованих в нього крихітних чіпів.

Для розробки рішення використовувався мова Python версії 3.7. Вибір в сторону Python зроблений завдяки простоті його застосування для задач обробки даних за допомогою математичних засобів.

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);
- відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Для реалізації застосовувалися такі допоміжні бібліотеки:

• scipy (зокрема scipy.signal і scipy.fft) - попередня обробка ісегментація сигналу;

• scikit-learn - реалізації класифікаторів, що використовуються для распознаванія P300;

• Qt for Python (PySide2) - призначений для користувача інтерфейс;

• pywin32 (обгортка над WinAPI для Python) - взаємодія з системой.

Для того, щоб перевірити, наскільки реалізоване рішення відповідає вимогам, а також для того, щоб серед аналогічних варіантів реалізації алгоритму розпізнавання P300 вибрати той, який дає найкращий результат на реальних даних, потрібно було провести ряд експериментів.

Для оцінки якості алгоритму розглянемо сценарій введення тексту з голови - перед нами стоїть завдання знайти серед 12 класів два релевантних. Спробуємо ввести вещественно значную метрику, що дозволяє нам порівняти варіанти, отримані на етапі розробки алгоритму розпізнавання P300.

Позначимо кількість випробувань в експерименті як total. нехай серед них у нас є correct випробувань, в результаті яких ми отримали результат, що співпадає з задуманим. Тоді ми можемо ввести метрику accuracy точності розпізнавання:

accuracy = correct. total

Результати експериментів усереднюються по всім випробовуваним.

Так як розробляється засіб передбачає використання в реальному часі, важливим параметром є час, який потрібен для однієї ітерації розпізнавання. Воно визначається однозначно параметрами сесії - для n класів і k повторень стимулів. Експеримент з оцінки якості розпізнавання P300 складається з двох фаз:

• Тренувальна сесія (100 випробувань) - від користувача потрібно вибрати конкретний вказаний клас. Дані тренувальної сесії використовуються для навчання класифікаторів (80 випробувань) та пошуку оптимальних значень гіперпараметров (20 випробувань).

• Безпосередньо експеримент (100 випробувань) - від користувача потрібно ввести запропонований текст з використанням рішення. Введений текст порівнюється з модельним за допомогою метрики accuracy.

Дані для експериментів були записані з залученням двох випробовуваних. Обидва випробовуваних були фізично здорові, не мали психічних відхилень і на момент запису даних не перебували в станах, які негативно позначаються на концентрації (недосип, перевтома і інші подібні стану).

Для якісного порівняння обраних методів (SVM, LDA) варіювалися наступні параметри.

• Метод видалення артефактів (PCA, ICA).

• Гіперпараметри - був проведений grid search (перебір) гіперпараметров. Вибиралися кількість відсікаються компонент ICA, частота зрізу смугового фільтра, довжина сегмента при сегментації та ін.

• Параметри сесії - їх зміна передбачає отримання нових даних. На відміну від зміни класифікатора або способів попередньої обробки, які не потребують повторної записи даних, при зміні кількості повторень стимулу, кількості класів та інтервалу між стимулами потрібно провести кілька різних експериментів. Однак, є можливість зімітувати зменшення кількості повторень стимулу, записавши дані з великим кількістю повторень і відкинувши кілька сегментів.

Спочатку алгоритм був протестований на даних [16] - вдалося досягти accuracy, рівній 91,0%, з допомогою такої комбінації:

• класифікатор - LDA;

• метод видалення артефактів - ICA;

• параметри сесії: TTI = 100 мс, 15 повторень стимулу (18 з на одну ітерацію введення). Зменшення кількості повторень стимулу до 5 зменшує час однієї ітерації введення до 6 с, але викликає зниження accuracy до значення 56.5%.

Аналогічна конфігурація, але з використанням нейронної мережі замість LDA дала результат 87.5% (для порівняння - при заміні бінарного класифікатора на випадковий, який повертає позитивну відповідь з ймовірністю 0.5, математичне очікування accuracy для всього експерименту виходить рівним 2.8%).

Також були випробувані підходи, які використовують SVM, але вони демонстрували неприйнятні результати на необроблених даних (менше 30%). Поліпшення якості для них вимагає додаткового кроку вилучення ознак. Спроби використання PCA замість ICA на етапі видалення артефактів також привели лише до погіршення якості.

На власних же даних, записаних в результаті експерименту, аналогічна конфігурація дала лише 57.5% accuracy - для запису набору даних [16] Застосовувалося медичне обладнання з великим кількістю датчиків, що, ймовірно, і є причиною того, що для Emotiv EPOC + були отримані більш погані результати.

ВИСНОВОК

В процесі виконання даної роботи були отримані наступні результати.

• Здійснено пошук рішень, що дозволяють здійснювати отримання даних ЕЕГ з Emotiv EPOC +. Для роботи з пристроєм була обрана бібліотека CyKit.

• Сформульовано ряд функціональних і не функціональних вимог до системи.

• Реалізовано алгоритм виявлення P300 в ЕЕГ, що складається з фаз предобработки, фільтрації і застосування бінарного класифікатора. Випробуваний ряд класифікаторів (SVM, LDA, повнозв'язні нейронні мережі) і методів попередньої обробки даних.

• З урахуванням вимог розроблений прототип рішення, що дозволяє здійснювати введення тексту і позиціонування курсору на комп'ютері під керуванням ОС Windows. Для цього була розроблена архітектура, в основу якої лягло відділення логіки реального часу від решти логіки. Був розроблений для користувача інтерфейс для відображення візуальних стимулів. Реалізація здійснена мовою Python з використанням фреймворку Qt.

Qt дозволяє запускати написане з його допомогою програмне забезпечення в більшості сучасних операційних систем шляхом простої компіляції програми для кожної системи без зміни вихідного коду. Включає в себе всі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Є повністю об'єктно-орієнтованим, розширюваним і підтримує техніку компонентного програмування.

Відмітна особливість - використання метаоб'єктного компілятора [⇨] - попередньої системи обробки вихідного коду. Розширення можливостей забезпечується системою плагінів, які можливо розміщувати безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна.

Комплектується візуальним середовищем розробки графічного інтерфейсу Qt Designer, що дозволяє створювати діалоги і форми в режимі WYSIWYG. У постачання Qt є Qt Linguist - графічна утиліта, що дозволяє спростити локалізацію і переклад програми на багато мов; і Qt Assistant - довідкова система Qt, що спрощує роботу з документацією по бібліотеці, а також дозволяє створювати кроссплатформенную довідку для розроблювального на основі Qt програмного забезпечення. Починаючи з версії 4.5.0 в комплект включена середовище розробки Qt Creator, яка включає редактор коду, довідку, графічні засоби Qt Designer і можливість налагодження додатків. Qt Creator може використовувати GCC або Microsoft VC ++ як компілятора і GDB як відладчика. Для Windows-версій бібліотека комплектується компілятором, заголовними і об'єктними файлами MinGW.

СПИСОК ЛІТЕРАТУРИ

1. Haider, Ali, Fazel-Rezai, Reza. Application of P300 Event-Related Potential in Brain-Computer Interface // Event-Related Potentials and Evoked Potentials / під ред. Phakkharawat Sittiprapaporn. - Rijeka: IntechOpen, 2017. - Гл. 2. - DOI:10. 5772 / intechopen. 69309. - URL:https://doi.org/10.5772/intechopen.69309.

2. A survey of signal processing algorithms in brain-computer interfaces based on electrical brain signals / Ali Bashashati [и др.] // Journal of Neural engineering. - 2007. - Т. 4, № 2. - R32.

3. Kumar , J. Satheesh, Bhuvaneswari, P. Analysis of Electroencephalography (EEG) Signals and Its Categorization-A Study // Procedia Engineering. - 2012. - Т. 38. - С. 2525-2536. - ISSN 1877-7058. - DOI: https://doi.org/10.1016/j.proeng.2012.

4. Lotte, Fabien. A Tutorial on EEG Signal Processing Techniques for Mental State Recognition in Brain-Computer Interfaces // Guide to Brain-Computer Music Interfacing / під ред. Eduardo Reck Miranda, Julien Castet. - Springer, 2014. - URL:https://hal.inria.fr/hal- 01055103.

5. Sur, S., Sinha, VK Event-related potential: An overview // Industrial psychiatry journal. - 2009. - 18 (1). - С. 70-73.

6. Polich,John, Kok, Albert. Cognitive and biological determinants of P300: an integrative review // Biological psychology. - 1995. - Т. 41, № 2. - С. 103-146.

7. P300 Waves for Single Subjects URL:http://www.cs.colostate.edu/eeg/data/json/doc/ tutorial / _build / html / p300_single_subject.html.

8. Wavelet analysis of oddball P300 / Tamer Demiralp [и др.] // International journal of psychophysiology. - 2001. - Т. 39, № 2/3. - С. 221-227.

9. Turnip, Arjon, Hong, Keum-Shik. Classifying mental activities from EEG- P300 signals using adaptive neural network // Int. J. Innov. Comp. Inf. Control. - 2012. - Т. 8, № 7. - С. 5839-5850.

10. Lakshmi, M. Rajya, Prasad, TV, Prakash, V. Chandra. Survey on EEG signal processing methods // International Journal of Advanced Research in Computer Science and Software Engineering. - 2014. - Т. 4, № 1.

11. P300 detection based on EEG shape features / Montserrat Alvarado- Gonz'alez [и др.] // Computational and mathematical methods in medicine. - 2016. - Т. 2016.

12. Al-Ani, Tarik, Trad, Dalila. Signal Processing and Classification Approaches for Brain-Computer Interface // / під ред. Vernon S. Somerset. - InTech, 01.2010. - ISBN 978-953-7619-58-9. - DOI:10. 5772/7032.

13. Roy , Vandana, Shukla, Shailja. A Survey on Artifacts Detection Techniques for Electro- Encephalography (EEG) Signals // International Journal of Multimedia and Ubiquitous Engineering. - 2015. - Т. 10, № 3. - С. 425-442.

14. Tsuda, Mineyuki, Lang, Yankun, Wu, Haiyuan. Analysis and Identification of the EEG Signals from Visual Stimulation // Procedia Computer Science. - 2014. - Т. 35. - С. 1292-1299. - ISSN 1877- 0509. - DOI: https://doi.org/10.1016/j.procs.2014.08.229. - URL:http: / / www. sciencedirect. com / science / article / pii / S1877050914011946 ; Knowledge-Based and Intelligent Information & Engineering Systems 18th Annual Conference, KES 2014 Gdynia, Poland,September 2014 Proceedings.

15. An experiment of lie detection based EEG-P300 classified by SVM algorithm / Artha Simbolon [и др.] //. - 10.2015. - DOI:10. 1109 / ICACOMIT.2015.7440177.

16. BCI Competition III. - URL:http://www.bbci.de/competition/ii/.

17. Rakotomamonjy, Alain, Guigue, Vincent. BCI competition III: dataset II-ensemble of SVMs for BCI P300 speller // IEEE transactions on biomedical engineering. - 2008. - Т. 55, № 3. - С. 1147-1154.

18. A boosting approach to P300 detection with application to brain-computerinterfaces / Ulrich Hoffmann [и др.] // Conference Proceedings. 2nd International IEEE EMBS Conference on Neural Engineering, 2005. - IEEE. 2005. - С. 97-100.

19.EmotivPRO - URL: https://www.emotiv.com/emotivpro/.

20. Emotiv Community SDK - URL:https://github.com/Emotiv/community-sdk.

ДОДАТОК А

```
import asyncio
import logging
import signal
from argparse import ArgumentParser
from model import ConcretePreprocessingStrategy, LDAModel
from realtime import Session
def main():
    logging.basicConfig(level=logging.WARNING)
    args = parse_args()
    preproc_strategy = ConcretePreprocessingStrategy()
    model = LDAModel()
    session = Session(args, preproc_strategy, model)
    loop = asyncio.get_event_loop()
    try:
        loop.add_signal_handler(signal.SIGINT, session.stop)
        loop.add_signal_handler(signal.SIGTERM, session.stop)
    except NotImplementedError:
        pass
    try:
        loop.run_until_complete(session.run())
    except KeyboardInterrupt:
        session.stop()
    finally:
        loop.close()
```

```
def parse_args():
    parser = ArgumentParser()
    parser.add_argument('cykit_address', help='IP address CyKit is located at')
    parser.add_argument('cykit_port', help='Port CyKit server listens on', type=int)
    parser.add_argument('interaction_port', help='Port interaction server listens on',
type=int)
    parser.add_argument('tti', help='Target-to-target interval',
                        type=float, nargs='?', default=0.3)
    parser.add_argument('num_repetitions', help='Number of repetitions of a
stimulus',
                        type=int, nargs='?', default=5)
    parser.add_argument('highlight_time', help='Highlight time in training mode',
                        type=float, nargs='?', default=5.0)
    parser.add_argument('segment_duration', help='Segment duration in samples
(after preprocessing)',
                        type=int, nargs='?', default=128)
    parser.add_argument('delay_between_iters', help='Delay between iterations',
                        type=float, nargs='?', default=1.0)
    return parser.parse_args()
if __name__ == '__main__':
    main()
import sys
from PySide2.QtCore import Qt, QProcess, QSettings
from PySide2.QtGui import QIcon, QPixmap, QTextCursor
from PySide2.QtWidgets import (
    QAction, QActionGroup, QApplication, QLabel, QMenu, QSystemTrayIcon,
QTextBrowser)
```

```
import app
from app.realtime_interaction import InteractionServer
from app.preferences import PreferencesDialog
from app.ui import KeyboardUI
logger = logging.getLogger(__name__)
class App(QApplication):
    def __init__(self, argv):
        super().__init__(argv)
        self._create_tray_icon()
        self._create_ui()
        self._create_interaction_server()
        self._session = None
    def open_preferences(self):
        prefs_dialog = PreferencesDialog()
        prefs_dialog.exec()
    def _mode_changed(self):
        action = self._mode_group.checkedAction()
        if action == self._mode_off:
            self._stop_session()
        elif action == self._mode_enabled:
            self._interaction_server.train = False
            self._start_session()
        elif action == self._mode_training:
            self._interaction_server.train = True
            self._start_session()
```

```
def _start_session(self):
    if self._session is not None:
        return
    self._session = QProcess(self)
    self._session.finished.connect(self._session_ended)
    self._session.readyReadStandardOutput.connect(self._log_append_stdout)
    self._session.readyReadStandardError.connect(self._log_append_stderr)
    settings = QSettings()
    self._session.start(sys.executable, [
        'run_session.py',
        settings.value('CyKitAddress', app.DEFAULT_CYKIT_ADDRESS),
        str(settings.value('CyKitPort', app.DEFAULT_CYKIT_PORT)),
        str(self._interaction_server.port)
    ])

def _stop_session(self):
    if self._session is not None:
        self._session.close()

# TODO: Handle non-null exit codes
def _session_ended(self):
    self._session = None
    self._mode_off.setChecked(True)

def _log_append_stdout(self):
```

```
        process = self.sender()
        self._log_window.moveCursor(QTextCursor.End)

self._log_window.insertPlainText(process.readAllStandardOutput().data().decode('
utf-8'))
        self._log_window.moveCursor(QTextCursor.End)

    def _log_append_stderr(self):
        process = self.sender()
        self._log_window.moveCursor(QTextCursor.End)

self._log_window.insertPlainText(process.readAllStandardError().data().decode('u
tf-8'))
        self._log_window.moveCursor(QTextCursor.End)

    def _select_letter(self, letter):
        self._letter_ui.setText(letter)

    def _create_tray_icon(self):
        menu = QMenu()

        self._mode_group = QActionGroup(menu)
        self._mode_group.triggered.connect(self._mode_changed)

        self._mode_off = QAction("&Off", parent=menu)
        self._mode_off.setCheckable(True)
        self._mode_off.setChecked(True)
```

```
self._mode_group.addAction(self._mode_off)
menu.addAction(self._mode_off)

self._mode_enabled = QAction("&Enabled", parent=menu)
self._mode_enabled.setCheckable(True)
self._mode_group.addAction(self._mode_enabled)
menu.addAction(self._mode_enabled)

self._mode_training = QAction("&Training mode", parent=menu)
self._mode_training.setCheckable(True)
self._mode_group.addAction(self._mode_training)
menu.addAction(self._mode_training)

menu.addSeparator()
menu.addAction("&Preferences", self.open_preferences)
menu.addSeparator()
menu.addAction("E&xit", self.exit)

pixmap = QPixmap(32, 32)
pixmap.fill(Qt.white)
icon = QIcon(pixmap)

self._tray_icon = QSystemTrayIcon(parent=self)
self._tray_icon.setContextMenu(menu)
self._tray_icon.setIcon(icon)
```

```
        self._tray_icon.show()

    def _create_ui(self):
        self._keyboard_ui = KeyboardUI()
        self._keyboard_ui.show()

        # TODO: Get rid of this in favor of os_interaction
        self._letter_ui = QLabel("-")
        self._letter_ui.setWindowTitle("Selected letter")
        self._letter_ui.setStyleSheet('font-size: 72pt')
        self._letter_ui.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self._letter_ui.setGeometry(600, 0, 100, 100)
        self._letter_ui.show()

        # TODO: Replace with more user-friendly log
        self._log_window = QTextBrowser()
        self._log_window.setWindowTitle("Session Log")
        self._log_window.setGeometry(700, 0, 500, 500)
        self._log_window.show()

    def _create_interaction_server(self):
        self._interaction_server = InteractionServer(self)

self._interaction_server.keyboard_flash_row.connect(self._keyboard_ui.flash_row)

self._interaction_server.keyboard_flash_col.connect(self._keyboard_ui.flash_col)
```

```
self._interaction_server.keyboard_highlight_letter.connect(self._keyboard_ui.highl
ight_letter)
        self._interaction_server.keyboard_select_letter.connect(self._select_letter)
import json
import struct


from PySide2.QtCore import QObject, Signal
from PySide2.QtNetwork import QHostAddress, QTcpServer


LENGTH_STRUCT = struct.Struct('N')



class InteractionServer(QObject):
    keyboard_flash_row = Signal(int)
    keyboard_flash_col = Signal(int)
    keyboard_highlight_letter = Signal(str)
    keyboard_select_letter = Signal(str)
    mouse_flash_class = Signal(int)
    mouse_highlight_class = Signal(int)
    mouse_select_class = Signal(int)


    def __init__(self, parent=None):
        super().__init__(parent)


        self._tcp_server = QTcpServer(self)
```

```
        self._tcp_server.listen(QHostAddress('localhost'))
        self._tcp_server.newConnection.connect(self._handle_new_connection)

        self.port = self._tcp_server.serverPort()
        self.train = False

    def _handle_new_connection(self):
        connection = self._tcp_server.nextPendingConnection()
        connection.readyRead.connect(self._handle_ready_read)

    def _handle_ready_read(self):
        socket = self.sender()
        while not socket.atEnd():
            length =
LENGTH_STRUCT.unpack(socket.read(LENGTH_STRUCT.size).data())[0]
            payload = socket.read(length).data()
            message = json.loads(payload.decode('utf-8'))
            self._handle_message(socket, message)

    def _send_message(self, socket, message):
        payload = json.dumps(message).encode('utf-8')
        socket.write(LENGTH_STRUCT.pack(len(payload)) + payload)
        socket.flush()

    def _handle_message(self, socket, message):
```

```
        if message['action'] == 'request_config':
            self._send_message(socket, {
                'mode': 'keyboard',
                'train': self.train
            })
        elif message['action'] == 'signal':
            signal = getattr(self, message['signal'])
            signal.emit(*message['args'])
import json
from abc import ABC, abstractmethod
from collections import defaultdict
from datetime import datetime

import joblib
import numpy as np
from matplotlib.figure import Figure
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

import model



def average_segments(stimuli, segments):
    d = defaultdict(list)
```

```
    for stimulus, segment in zip(stimuli, segments):
        d[stimulus].append(segment[1])
    return {k: np.mean(np.array(v), axis=0).flatten() for k, v in d.items()}


class Model(ABC):
    @abstractmethod
    def train_iteration(self, stimuli, segments, target):
        raise NotImplementedError

    @abstractmethod
    def get_probabilities(self, stimuli, segments):
        raise NotImplementedError


class RecordModel(Model):
    SCALE_FACTOR = 4.0

    def train_iteration(self, stimuli, segments, target):
        filename = "train_" + self._get_timestamp()
        self._plot(filename, stimuli, segments, target)
        self._dump(filename, {"stimuli": stimuli, "segments": segments, "target":
target})

    def get_probabilities(self, stimuli, segments):
```

```
        filename = "work_" + self._get_timestamp()
        self._plot(filename, stimuli, segments, [])
        self._dump(filename, {"stimuli": stimuli, "segments": segments})
        return {x: 0.0 for x in stimuli}


    @staticmethod
    def _plot(filename, stimuli, segments, target):
        rows, cols = RecordModel._get_rows_cols(len(stimuli))
        fig = Figure(figsize=(RecordModel.SCALE_FACTOR * cols,
RecordModel.SCALE_FACTOR * rows))
        axs = fig.subplots(rows, cols)


        for i, (stimulus, (start, data)) in enumerate(zip(stimuli, segments)):
            args = [str(stimulus), str(start)]
            if stimulus in target:
                args.append("relevant")
            caption = ', '.join(args)


            row, col = i // cols, i % cols
            axs[row, col].set_title(caption)
            axs[row, col].plot(data)


        directory = model.CONFIG_DIRECTORY / "images"
        directory.mkdir(parents=True, exist_ok=True)
        fig.savefig(directory / "{}.png".format(filename))
```

```
    @staticmethod
    def _dump(filename, data):
        directory = model.CONFIG_DIRECTORY / "data"
        directory.mkdir(parents=True, exist_ok=True)
        path = directory / "{}.txt".format(filename)
        with open(path, 'w') as f:
            json.dump(data, f)

    @staticmethod
    def _get_timestamp():
        return datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

    @staticmethod
    def _get_rows_cols(n):
        rows, cur = 1, 2
        while cur * cur <= n:
            if n % cur == 0:
                rows = cur
            cur += 1
        return rows, n // rows


class LDAModel(RecordModel):
    def __init__(self):
```

```
        directory = model.CONFIG_DIRECTORY / "models"
        directory.mkdir(parents=True, exist_ok=True)
        self.filename = directory / "lda_model.joblib"

        if self.filename.exists():
            self._clf = joblib.load(self.filename)
        else:
            self._clf = make_pipeline(
                StandardScaler(),
                LinearDiscriminantAnalysis()
            )

        self._X, self._y = [], []

    def train_iteration(self, stimuli, segments, target):
        super().train_iteration(stimuli, segments, target)

        for stimulus, x in average_segments(stimuli, segments).items():
            self._X.append(x)
            self._y.append(int(stimulus in target))

    def get_probabilities(self, stimuli, segments):
        parent_proba = super().get_probabilities(stimuli, segments)

        if self._X:
```

```
            self._clf.fit(self._X, self._y)
            joblib.dump(self._clf, str(self.filename))
            self._X, self._y = [], []

        probs = {}
        for stimulus, x in average_segments(stimuli, segments).items():
            probs[stimulus] = self._clf.predict_proba([x])[0, 1]
        return probs
from abc import ABC

import numpy as np
import scipy.signal
from sklearn.decomposition import FastICA


class PreprocessingStrategy(ABC):
    def __init__(self, batch_size=16, sample_rate=128):
        self.batch_size = batch_size
        self.sample_rate = sample_rate

    def preprocess_batch(self, batch):
        return batch


class ConcretePreprocessingStrategy(PreprocessingStrategy):
```

```
    def __init__(self, bandpass_lcf=0.4, bandpass_hcf=30.0, subsample_rate=2,
num_removed_comps=4):
        super().__init__(batch_size=512)

        self.bandpass_lcf = bandpass_lcf
        self.bandpass_hcf = bandpass_hcf
        self.subsample_rate = subsample_rate
        self.num_removed_comps = num_removed_comps

    def preprocess_batch(self, batch):
        batch = np.array(batch)

        res = self._bandpass(batch, self.bandpass_lcf, self.bandpass_hcf)
        res = self._subsample(res, self.subsample_rate)
        res = self._ica(res, self.num_removed_comps)

        return res.tolist()

    def _bandpass(self, batch, lcf, hcf):
        nyquist_freq = self.sample_rate / 2
        b, a = scipy.signal.butter(5, [lcf / nyquist_freq, hcf / nyquist_freq],
btype='band')
        res = scipy.signal.lfilter(b, a, batch)
        return res

    def _subsample(self, batch, k=2):
```

```
        return batch[::k].repeat(k, axis=0)


    def _ica(self, batch, num_comps=4):
        ica = FastICA(n_components=batch.shape[1], max_iter=300)
        comps = ica.fit_transform(batch)
        m = (-np.abs(comps)).min(axis=0)
        selected = np.argsort(m)[:num_comps]
        comps[:, selected] = 0
        return ica.inverse_transform(comps)
import asyncio
from concurrent.futures.thread import ThreadPoolExecutor

from model import PreprocessingStrategy



class Preproc:
    def __init__(self, preprocessing_strategy: PreprocessingStrategy):
        self.preprocessing_strategy = preprocessing_strategy
        self._segments = set()
        self._counter = 0

    async def run(self, cykit_client, executor=None):
        batch = []
        batch_start = self._counter
```

```
    async for sample in cykit_client:
        batch.append(sample)
        self._counter += 1
        if len(batch) < self.preprocessing_strategy.batch_size:
            continue

        with ThreadPoolExecutor() as executor:
            loop = asyncio.get_event_loop()
            preprocessed_batch = await loop.run_in_executor(
                executor, self.preprocessing_strategy.preprocess_batch, batch)
        for segment in self._segments:
            segment.append_batch(preprocessed_batch, batch_start)

        batch = []
        batch_start = self._counter

async def get_segment(self, duration=128):
    segment = PartialSegment(duration, self._counter)

    self._segments.add(segment)
    data = await segment.get_complete_data()
    self._segments.discard(segment)

    return segment.start, data
```

```
class PartialSegment:
    def __init__(self, duration, start):
        self.duration = duration
        self.start = start

        self._first_batch_start = None
        self._samples = []
        self._queue = asyncio.Queue()

    async def get_complete_data(self):
        return await self._queue.get()

    def append_batch(self, batch, batch_start):
        self._samples += batch

        if self._first_batch_start is None:
            self._first_batch_start = batch_start
        offset = self.start - self._first_batch_start
        if len(self._samples) - offset < self.duration:
            return

        self._samples = self._samples[offset:(offset + self.duration)]
        self._queue.put_nowait(self._samples)
import asyncio
```

```
import random
from argparse import Namespace
from concurrent.futures import ThreadPoolExecutor

from model import PreprocessingStrategy, Model
from realtime.acquisition import connect_to_cykit
from realtime.app_interaction import connect_to_app
from realtime.preprocessing import Preproc

LETTERS = ['ABCDEF', 'GHIJKL', 'MNOPQR', 'STUVWX', 'YZ0123', '456789']
NUM_MOUSE_CLASSES = 5


class Session:
    def __init__(self, args: Namespace, preprocessing_strategy:
PreprocessingStrategy, model: Model):
        self.preprocessing_strategy = preprocessing_strategy
        self.model = model
        self.args = args
        self._cykit_client = None
        self._interaction_client = None
        self._preproc = Preproc(self.preprocessing_strategy)
        self._executor = None
        self._wait_for_executor()
```

```
    async def run(self):
        try:
            self._cykit_client = await connect_to_cykit(self.args.cykit_address,
self.args.cykit_port)
            self._interaction_client = await connect_to_app(self.args.interaction_port)
            await asyncio.gather(
                self._preproc.run(self._cykit_client, self._executor),
                self._run_session())

        finally:
            self.stop()

    def stop(self):
        if self._cykit_client is not None:
            self._cykit_client.stop()
        if self._interaction_client is not None:
            self._interaction_client.stop()

    async def _run_session(self):
        while True:
            meta = await self._interaction_client.request_config()
            if meta['mode'] == 'keyboard':
                await self._run_keyboard_iteration(meta['train'])
            elif meta['mode'] == 'mouse':
                await self._run_mouse_iteration(meta['train'])
```

```
            else:
                raise RuntimeError("Invalid mode")

    # TODO: Extract common code from this and _run_single_class_iteration
    async def _run_keyboard_iteration(self, train=False):
        target_row, target_col = None, None
        if train:
            target_row = random.randrange(0, len(LETTERS))
            target_col = random.randrange(0, len(LETTERS[0]))
            target_letter = LETTERS[target_row][target_col]

asyncio.create_task(self._interaction_client.signal('keyboard_highlight_letter',
target_letter))
            await asyncio.sleep(self.args.highlight_time)


        stimuli = []
        for _ in range(self.args.num_repetitions):
            for i in range(len(LETTERS)):
                stimuli.append(('row', i))
            for i in range(len(LETTERS[0])):
                stimuli.append(('col', i))
        random.shuffle(stimuli)


        futures = []
        for stimulus_type, idx in stimuli:
            if stimulus_type == 'row':
```

```
            asyncio.create_task(self._interaction_client.signal('keyboard_flash_row',
idx))
        elif stimulus_type == 'col':
            asyncio.create_task(self._interaction_client.signal('keyboard_flash_col',
idx))
        fut =
asyncio.create_task(self._preproc.get_segment(duration=self.args.segment_duratio
n))
        futures.append(fut)
        await asyncio.sleep(self.args.tti)


    segments = await asyncio.gather(*futures)
    if train:
        target = [('row', target_row), ('col', target_col)]
        asyncio.create_task(self._train_iteration(stimuli, segments, target))
    else:
        probs = await self._get_probabilities(stimuli, segments)
        relevant_row = max(filter(lambda x: x[0] == 'row', probs),
key=probs.get)[1]
        relevant_col = max(filter(lambda x: x[0] == 'col', probs), key=probs.get)[1]
        relevant_letter = LETTERS[relevant_row][relevant_col]
        asyncio.create_task(self._interaction_client.signal('keyboard_select_letter',
relevant_letter))


    # During this time, any extra unprocessed packets will be ignored to avoid
latency
    await asyncio.sleep(self.args.delay_between_iters)
```

```
    async def _run_mouse_iteration(self, train=False):
        target_class = None
        if train:
            target_class = random.randrange(0, NUM_MOUSE_CLASSES)
            asyncio.create_task(self._interaction_client.signal('mouse_highlight_class',
target_class))
            await asyncio.sleep(self.args.highlight_time)

        stimuli = list(range(NUM_MOUSE_CLASSES)) * self.args.num_repetitions
        random.shuffle(stimuli)

        futures = []
        for idx in stimuli:
            asyncio.create_task(self._interaction_client.signal('mouse_flash_class',
idx))
            fut =
asyncio.create_task(self._preproc.get_segment(duration=self.args.segment_duratio
n))
            futures.append(fut)
            await asyncio.sleep(self.args.tti)

        segments = await asyncio.gather(*futures)
        if train:
            asyncio.create_task(self._train_iteration(stimuli, segments, [target_class]))
        else:
            probs = await self._get_probabilities(stimuli, segments)
            relevant_class = max(probs, key=probs.get)
```

```
            asyncio.create_task(self._interaction_client.signal('mouse_select_class',
relevant_class))


        await asyncio.sleep(self.args.delay_between_iters)


    async def _train_iteration(self, stimuli, segments, target):
        self._wait_for_executor()
        return await asyncio.get_event_loop().run_in_executor(
            self._executor, self.model.train_iteration, stimuli, segments, target)


    async def _get_probabilities(self, stimuli, segments):
        self._wait_for_executor()
        return await asyncio.get_event_loop().run_in_executor(
            self._executor, self.model.get_probabilities, stimuli, segments)


    def _wait_for_executor(self):
        if self._executor is not None:
            self._executor.shutdown(wait=True)
        self._executor = ThreadPoolExecutor()
```