UNIVERSITY OF PETROLEUM AND ENERGY STUDIES, DEHRADUN

TITLE: Electroencephalogram (EEG) based Brain Computer Interface (BCI)

PROJECT REPORT OF THE (MINOR PROJECT - 1) IN SEMESTER VI

S. No	Students Name	Roll No	Sap Id
1.	Shruti Tripathi	R2142210753	500094094
2.	Aditi Tiwari	R2142210045	500095658
3.	Vansh Santdasani	R2142210838	500093950

BACHELOR OF TECHNOLOGY, COMPUTER SCIENCE

Under the guidance of

Dr. Kingshuk Srivastava

School Computer Science (SOCS), Data Science Cluster, UPES

Bidholi Campus, Energy Acres, Dehradun – 248007

Table of Contents

List of Figures

Abstract

The field of Brain-Computer Interface (BCI) technology has witnessed remarkable advancements, offering transformative possibilities for human-machine interactions across diverse domains. This project aims to explore and showcase the vast applications of BCI technology through the implementation of various projects. The methodology involves a systematic approach, encompassing brainwave selection, EEG system exploration, continuous data streaming, closed-loop BCI architecture design, and rigorous validation. By omitting machine learning, the focus shifts towards demonstrating the versatility and practicality of BCI in real-time scenarios.

The projects undertaken in this study span a spectrum of applications, ranging from healthcare to entertainment, emphasizing the adaptability and potential impact of BCI technology. Through meticulous design, implementation, and evaluation, each project serves as a testament to the broad utility of BCI systems beyond traditional paradigms.

This abstract provides a glimpse into the comprehensive exploration of BCI applications, showcasing the feasibility of integrating brainwave signals into various real-world projects. The outcomes of this study contribute to the growing body of knowledge surrounding BCI technology, illustrating its potential to redefine how humans interact with technology and the world around them.

1. Introduction

1.1 Introduction

In the evolving landscape of human-computer interaction, the advent of Brain-Computer Interface (BCI) technology has emerged as a transformative force, promising unparalleled synergy between the human mind and technological systems. Rooted in interdisciplinary pursuits, BCI integrates principles from neuroscience, signal processing, and human factors to establish a conduit for direct communication between the human brain and external devices. This paper endeavors to explore and substantiate the myriad applications of BCI technology through the implementation of meticulously designed projects, thereby contributing to the discourse surrounding its practical utility.

The methodological underpinning of this exploration involves a deliberate emphasis on brainwave selection, EEG system optimization, and real-time data processing. Notably, the deliberate exclusion of machine learning complexities serves to illuminate the innate potential and versatility of BCI in a pragmatic context. The ensuing projects are poised to exemplify BCI's capability to transcend theoretical paradigms and seamlessly integrate into diverse real-world scenarios.

1.2 Aim of the Project

The primary aim of this project is to demonstrate the abilities of Brain Computer Interface (BCI) by implementing a maze game where the character is controlled and moved by the brain signals.

1.3 Technical Approach

1. Literature Review: Identify and gather relevant academic papers, books, articles, and online resources related to generative modeling.

2. Platform and Tool Recognition and Identification: Choose the most appropriate combination of platforms and tools based on project requirements, considering factors like scalability, community support, and integration capabilities.

3. Designing and Development of the Model: Define the architecture and structure of the library, including the training, and evaluation.

4. Testing and Optimization of the Model: Collect user feedback through usability testing to identify areas for improvement.

1.4 Significance of the work

This project is the perfect demonstration of application of EEG in machines and its abilities to control objects in real time.

In simple terms, our goal is to control the character in a maze using a person's brain waves. Our aim is to deliver a maze game which can be controlled using the focus- relaxation mechanism where when the user focuses the object moves right and when they relax it moves left. This concept uses the concept of different forms of waves and calibrating the frequencies based on every brain.

2. Project Proposal

2.1 Literature Review

We researched in the fields of BCI and EEG applications and these were some of our findings. In ‘Making Sense of Spatio-Temporal Preserving Representations for EEG-Based Human Intention Recognition’ by Dalin et. al.[1], they introduce two deep learning-based frameworks with novel spatio-temporal preserving representations of raw EEG streams to precisely identify human intentions.The developed models are further evaluated with a real-world brain typing BCI and achieve a recognition accuracy of 93% over five instruction intentions suggesting good generalization over different kinds of intentions and BCI systems.

In ‘Automatically Identified EEG Signals of Movement Intention Based on CNN Network (End-To-End)’ by Nahal et. al.[2] introduces a novel method for automatically categorizing two-class and three-class movement-intention situations utilizing EEG data. In the suggested technique, the raw EEG input is applied directly to a convolutional neural network (CNN) without feature extraction or selection. The suggested approach could be employed in BCI applications due to its high accuracy.

In ‘Exploration of computational methods for classification of movement intention during human voluntary movement from single trial EEG’ by Ou Bai et. al. [3] utilized one of the better combinations of ICA, PSD and SVM, the discrimination accuracy was as high as 75%. Further feature analysis showed that beta band EEG activity of the channels over the right sensorimotor cortex was most appropriate for discrimination of right and left hand movement intention.

In ‘Prediction of gait intention from pre-movement EEG signals: a feasibility study’ by Hasan et. al.[4] show that it is possible to achieve statistically similar intention detection accuracy using either only pre-movement EEG features or trans-movement EEG features. The classifier performance shows the potential of the proposed methodology to predict human movement intention exclusively from the pre-movement EEG signal to be applied in real-life prosthetic and neuro-rehabilitation systems.

We also looked into ‘A new method for accurate detection of movement intention from single channel EEG for online BCI’ by Mahmoodi et. al[5] to increase our knowledge on single channel EEG and ‘Exploring BCI Control in Smart Environments: Intention Recognition Via

EEG Representation Enhancement Learning' by Lin Yue et.al. [6] to learn about motor imagery using BCI.

2.2 Problem Statement

Despite significant strides, the widespread adoption of real-time EEG-based BCIs faces several hurdles:

1. **Limited Awareness:** Public understanding of BCI technology remains limited, hindering potential user adoption and investment. The vast array of applications and transformative possibilities remain largely unexplored by the public.
2. **Technical Refinement:** While significant advancements have been made, the accuracy, robustness, and adaptability of real-time BCI systems require further improvement. Individual brain variations and environmental noise pose challenges to reliable and seamless interaction.
3. **Accessibility Barriers:** Cost, complexity, and the lack of user-friendly interfaces can create significant barriers to entry, excluding individuals who could benefit most from BCI technology.

2.3 Methodology

1. Literature Review and Brainwave Selection:

a. Conduct an extensive review of literature to identify the most relevant brainwave types for the targeted BCI application, considering factors such as cognitive tasks, user intentions, and signal interpretability.

b. Explore the physiological and cognitive implications of different brainwave frequencies, including alpha, beta, theta, and gamma waves.

c. Select specific brainwave signals based on their relevance to the intended BCI control tasks, considering existing research and established principles in neuroscience.

2. EEG System Exploration and Evaluation:

a. Understanding the EEG hardware and the different plots it produces. The relation between different plots and parameters that are being shown and what exactly is important to us

b. For our project, we are exploring and evaluating OpenBCI hardware and using the OpenBCI- GUI to collect and stream data.

3. Continuous Data Streaming and Pre-processing:

a. Implement real-time data streaming techniques to facilitate the continuous flow of raw EEG data from the acquisition system.

b. Employ pre-processing algorithms for noise reduction, baseline correction, and artifact removal while preserving the inherent characteristics of the selected brainwave signals.

c. Validate the effectiveness of pre-processing steps through quantitative and qualitative analysis, emphasizing real-time applicability and ensuring data integrity.

4. Closed-Loop BCI Architecture Design:

a. Develop a closed-loop BCI system architecture, incorporating the selected brainwave signals and real-time data streaming.

b. Establish a feedback loop between the user's brain activity and the controlled device or environment to enable seamless interaction.

c. Design and implement different output interfaces for real-time user engagement, such as cursor movement, robotic control, or virtual reality interactions.

5. System Integration and Validation:

a. Integrate all components into a cohesive BCI system, ensuring seamless communication between the EEG system, data processing modules, and output interfaces.

b. Conduct rigorous testing and validation procedures, assessing real-time performance, accuracy, and user satisfaction.

c. Iterate and refine the system based on empirical findings, user feedback, and comparative analyses with existing benchmarks or alternative methodologies.

6. Documentation and Dissemination:

a. Document the entire BCI development process, including methodologies, algorithms, and system architecture, for future reference and reproducibility.

b. Disseminate research findings through academic publications, conference presentations, and open-access platforms to contribute to the broader BCI research community.

2.4 Solution to the problem

Create an interactive maze game using OpenBCI hardware and python language. The game demonstrates in the simplest ways how brain signals can be utilized to achieve various tasks and that moving an object left or right is just one of the most simplest tasks that can be achieved using this technology.

3. Project Description

3.1 Reference Algorithm

We take reference from various papers and github repositories to try and understand the various approaches taken. One common approach we found was the use of Steady state visually evoked potential (SSVEP). We have decided to take a different approach to see the difference between our results.

3.2 Characteristic of Data

- All data is being captured and processed in real-time so there is no need for storing and managing any sort of data. We are processing the data in realtime and making decisions.

3.3 Process Flow Diagram

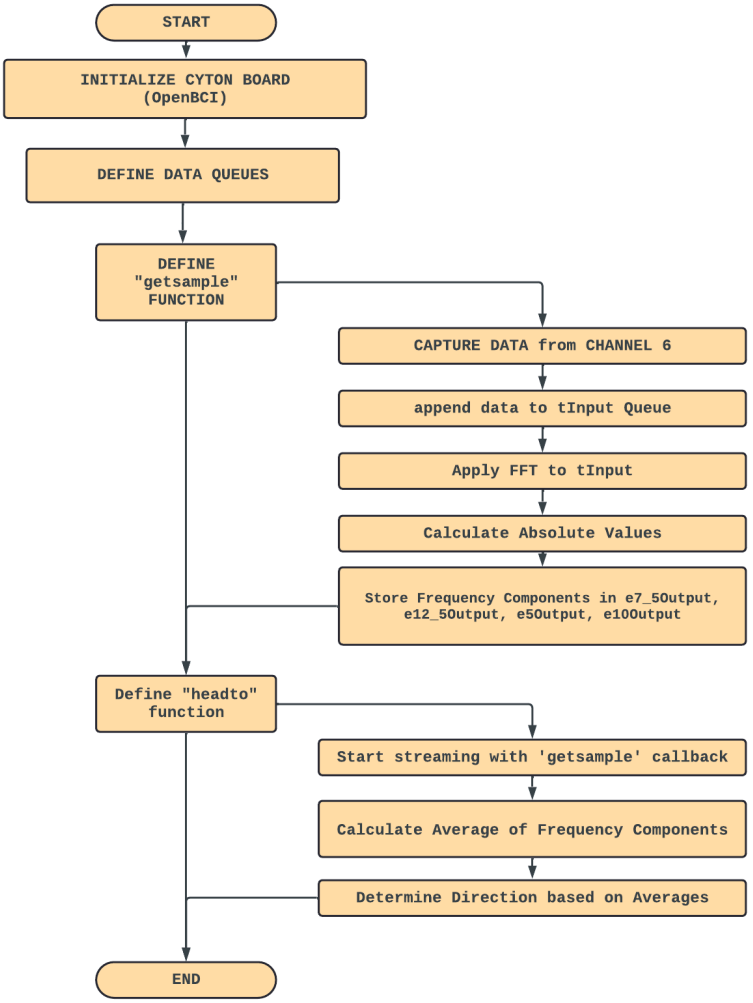

Fig 1.1 Real-time Data Processing from open_bci_v3 and brain_improved.py

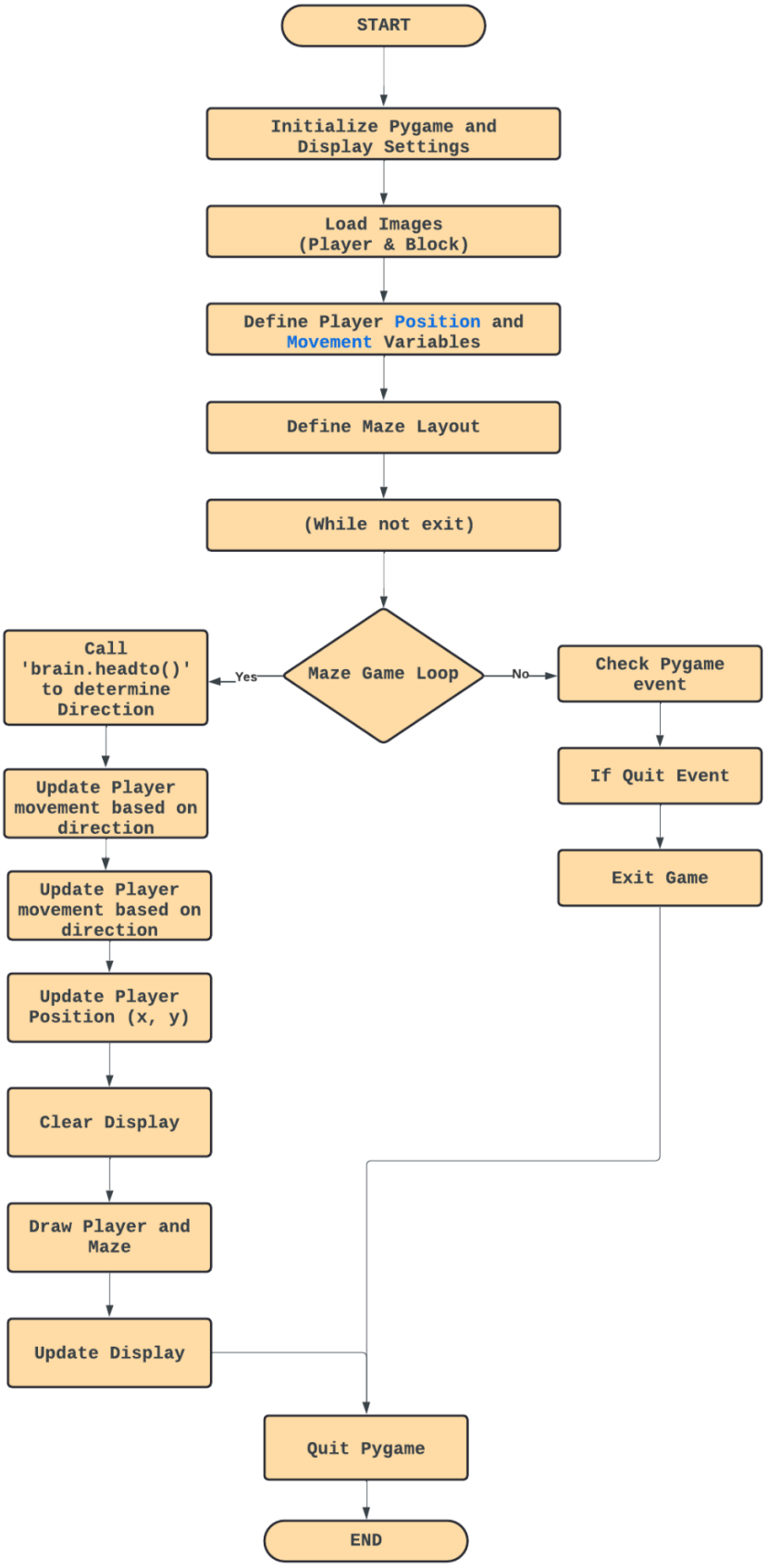

Fig 1.2 Flowchart for maze.py using open_bci_v3 and brain

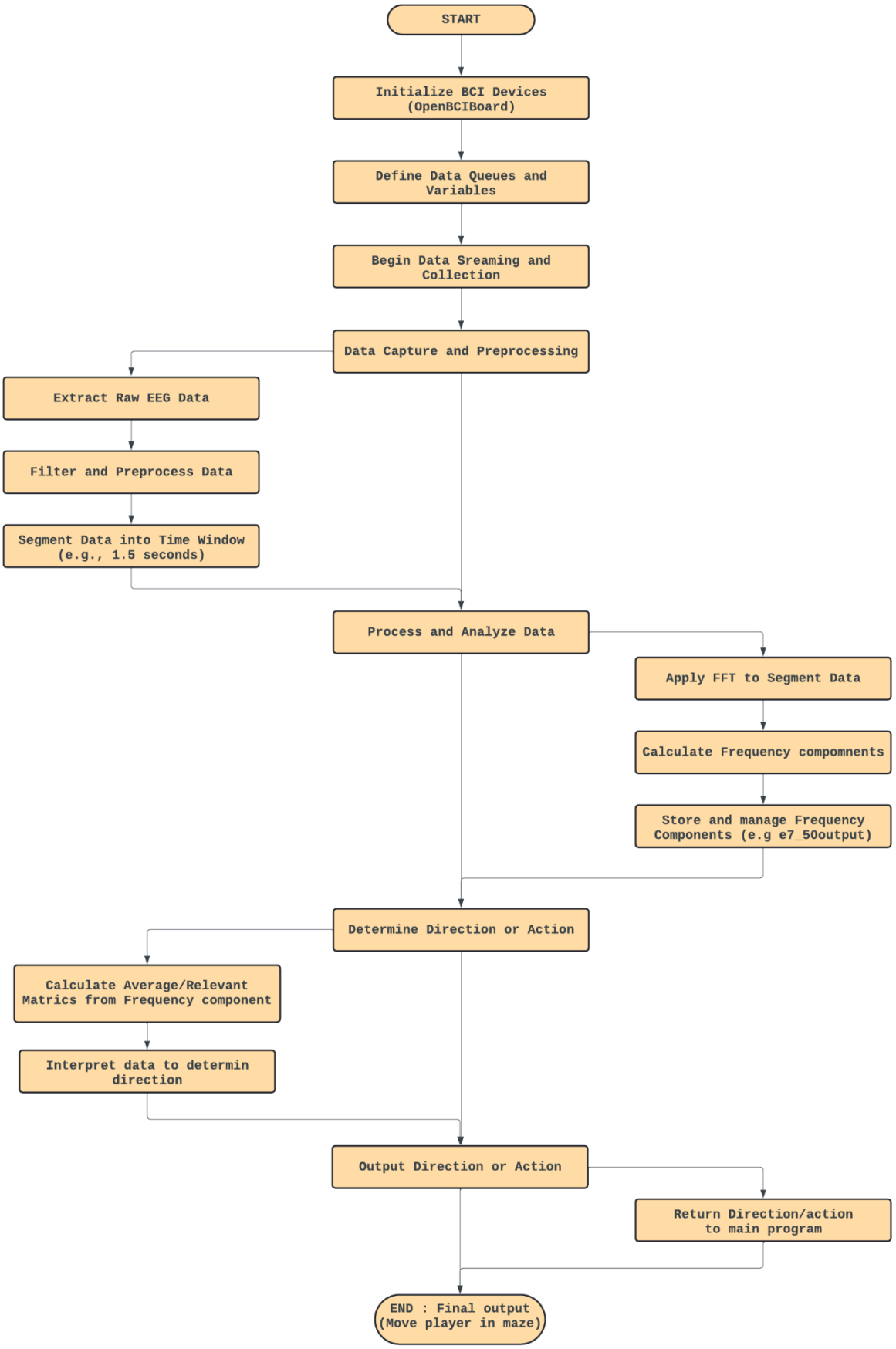

Fig 1.3 General Data Flow Structure

3.4 Program Code and Findings

3.4.1. Open_bci_v3.py (To set up and create a connection with the EEG device)

"""
Core OpenBCI object for handling connections and samples from the board.

EXAMPLE USE:

def handle_sample(sample):
  print(sample.channels)

board = OpenBCIBoard()
board.print_register_settings()
board.start(handle_sample)

NOTE: If daisy modules is enabled, the callback will occur every two samples, hence "packet_id" will only contain even numbers. As a side effect, the sampling rate will be divided by 2.

FIXME: at the moment we can just force daisy mode, do not check that the module is detected.

"""
```
import serial
import struct
import numpy as np
import time
import timeit
import atexit
import logging
import threading
import sys
import pdb
import glob

SAMPLE_RATE = 250.0  # Hz
START_BYTE = 0xA0  # start of data packet
END_BYTE = 0xC0  # end of data packet
ADS1299_Vref = 4.5  #reference voltage for ADC in ADS1299.  set by its hardware
ADS1299_gain = 24.0  #assumed gain setting for ADS1299.  set by its Arduino code
```

```
scale_fac_uVolts_per_count =
ADS1299_Vref/float((pow(2,23)-1))/ADS1299_gain*1000000.
scale_fac_accel_G_per_count = 0.002 /(pow(2,4)) #assume set to +/4G, so 2 mG
'''
#Commands for in SDK http://docs.openbci.com/software/01-Open BCI_SDK:

command_stop = "s";
command_startText = "x";
command_startBinary = "b";
command_startBinary_wAux = "n";
command_startBinary_4chan = "v";
command_activateFilters = "F";
command_deactivateFilters = "g";
command_deactivate_channel = {"1", "2", "3", "4", "5", "6", "7", "8"};
command_activate_channel = {"q", "w", "e", "r", "t", "y", "u", "i"};
command_activate_leadoffP_channel = {"!", "@", "#", "$", "%", "^", "&", "*"};  //shift + 1-8
command_deactivate_leadoffP_channel = {"Q", "W", "E", "R", "T", "Y", "U", "I"};   //letters
(plus shift) right below 1-8
command_activate_leadoffN_channel = {"A", "S", "D", "F", "G", "H", "J", "K"}; //letters
(plus shift) below the letters below 1-8
command_deactivate_leadoffN_channel = {"Z", "X", "C", "V", "B", "N", "M", "<"};
//letters (plus shift) below the letters below the letters below 1-8
command_biasAuto = "`";
command_biasFixed = "~";
'''

class OpenBCIBoard(object):
  """

  Handle a connection to an OpenBCI board.

  Args:
        port: The port to connect to.
        baud: The baud of the serial connection.
        daisy: Enable or disable daisy module and 16 chans readings
  """

  def __init__(self, port=None, baud=115200, filter_data=True,
        scaled_output=True, daisy=False, log=True, timeout=None):
        self.log = log # print_incoming_text needs log
        self.streaming = False
        self.baudrate = baud
        self.timeout = timeout
        if not port:
```

```
        port = self.find_port()
        self.port = port
        print("Connecting to V3 at port %s" %(port))
        self.ser = serial.Serial(port= 'COM5', baudrate = baud, timeout=100)

        print("Serial established...")

        time.sleep(2)
        #Initialize 32-bit board, doesn't affect 8bit board
        self.ser.write(b'v');


        #wait for device to be ready
        time.sleep(1)
        self.print_incoming_text()

        self.streaming = False
        self.filtering_data = filter_data
        self.scaling_output = scaled_output
        self.eeg_channels_per_sample = 8 # number of EEG channels per sample *from the
board*
        self.aux_channels_per_sample = 3 # number of AUX channels per sample *from the
board*
        self.read_state = 0
        self.daisy = daisy
        self.last_odd_sample = OpenBCISample(-1, [], []) # used for daisy
        self.log_packet_count = 0
        self.attempt_reconnect = False
        self.last_reconnect = 0
        self.reconnect_freq = 5
        self.packets_dropped = 0

        #Disconnects from board when terminated
        atexit.register(self.disconnect)

  def getSampleRate(self):
        if self.daisy:
        return SAMPLE_RATE/2
        else:
        return SAMPLE_RATE

  def getNbEEGChannels(self):
        if self.daisy:
        return self.eeg_channels_per_sample*2
```

```
        else:
        return self.eeg_channels_per_sample

  def getNbAUXChannels(self):
        return  self.aux_channels_per_sample

  def start_streaming(self, callback, lapse=-1):
        """
        Start handling streaming data from the board. Call a provided callback
        for every single sample that is processed (every two samples with daisy module).

        Args:
        callback: A callback function -- or a list of functions -- that will receive a single
argument of the
        OpenBCISample object captured.
        """
        if not self.streaming:
        self.ser.write(b'b')
        self.streaming = True

        start_time = timeit.default_timer()

        # Enclose callback funtion in a list if it comes alone
        if not isinstance(callback, list):
        callback = [callback]


        #Initialize check connection
        self.check_connection()

        while self.streaming:

        # read current sample
        sample = self._read_serial_binary()
        # if a daisy module is attached, wait to concatenate two samples (main board + daisy)
before passing it to callback
        if self.daisy:
        # odd sample: daisy sample, save for later
        if ~sample.id % 2:
        self.last_odd_sample = sample
        # even sample: concatenate and send if last sample was the fist part, otherwise drop
the packet
        elif sample.id - 1 == self.last_odd_sample.id:
```

```
        # the aux data will be the average between the two samples, as the channel samples
themselves have been averaged by the board
        avg_aux_data = list((np.array(sample.aux_data) +
np.array(self.last_odd_sample.aux_data))/2)
        whole_sample = OpenBCISample(sample.id, sample.channel_data +
self.last_odd_sample.channel_data, avg_aux_data)
        for call in callback:
        call(whole_sample)
        else:
        for call in callback:
        call(sample)

        if(lapse > 0 and timeit.default_timer() - start_time > lapse):
        self.stop();
        if self.log:
        self.log_packet_count = self.log_packet_count + 1;


    """
        PARSER:
        Parses incoming data packet into OpenBCISample.
        Incoming Packet Structure:
        Start Byte(1)|Sample ID(1)|Channel Data(24)|Aux Data(6)|End Byte(1)
        0xA0|0-255|8, 3-byte signed ints|3 2-byte signed ints|0xC0

    """
    def _read_serial_binary(self, max_bytes_to_skip=3000):
        def read(n):
        b = self.ser.read(n)
        if not b:
        self.warn('Device appears to be stalled. Quitting...')
        sys.exit()
        raise Exception('Device Stalled')
        sys.exit()
        return '\xFF'
        else:
        return b

        for rep in range(max_bytes_to_skip):

        #---------Start Byte & ID---------
        if self.read_state == 0:

        b = read(1)
```

```
if struct.unpack('B', b)[0] == START_BYTE:
if(rep != 0):
self.warn('Skipped %d bytes before start found' %(rep))
rep = 0;
packet_id = struct.unpack('B', read(1))[0] #packet id goes from 0-255
log_bytes_in = str(packet_id);

self.read_state = 1

#---------Channel Data---------
elif self.read_state == 1:
channel_data = []
for c in range(self.eeg_channels_per_sample):

#3 byte ints
literal_read = read(3)

unpacked = struct.unpack('3B', literal_read)
log_bytes_in = log_bytes_in + '|' + str(literal_read);

#3byte int in 2s compliment
if (unpacked[0] >= 127):
pre_fix = bytes(bytearray.fromhex('FF'))
else:
pre_fix = bytes(bytearray.fromhex('00'))


literal_read = pre_fix + literal_read;

#unpack little endian(>) signed integer(i) (makes unpacking platform independent)
myInt = struct.unpack('>i', literal_read)[0]

if self.scaling_output:
channel_data.append(myInt*scale_fac_uVolts_per_count)
else:
channel_data.append(myInt)

self.read_state = 2;

#---------Accelerometer Data---------
elif self.read_state == 2:
aux_data = []
for a in range(self.aux_channels_per_sample):
```

```
        #short = h
        acc = struct.unpack('>h', read(2))[0]
        log_bytes_in = log_bytes_in + '|' + str(acc);

        if self.scaling_output:
        aux_data.append(acc*scale_fac_accel_G_per_count)
        else:
        aux_data.append(acc)

        self.read_state = 3;
        #---------End Byte---------
        elif self.read_state == 3:
        val = struct.unpack('B', read(1))[0]
        log_bytes_in = log_bytes_in + '|' + str(val);
        self.read_state = 0 #read next packet
        if (val == END_BYTE):
        sample = OpenBCISample(packet_id, channel_data, aux_data)
        self.packets_dropped = 0
        return sample
        else:
        self.warn("ID:<%d> <Unexpected END_BYTE found <%s> instead of <%s>"
        %(packet_id, val, END_BYTE))
        logging.debug(log_bytes_in);
        self.packets_dropped = self.packets_dropped + 1

"""

Clean Up (atexit)

"""
def stop(self):
        print("Stopping streaming...\nWait for buffer to flush...")
        self.streaming = False
        self.ser.write(b's')
        if self.log:
        logging.warning('sent <s>: stopped streaming')

def disconnect(self):
        if(self.streaming == True):
        self.stop()
        if (self.ser.isOpen()):
        print("Closing Serial...")
        self.ser.close()
```

```
        logging.warning('serial closed')


"""

        SETTINGS AND HELPERS

"""
def warn(self, text):
        if self.log:
        #log how many packets where sent succesfully in between warnings
        if self.log_packet_count:
        logging.info('Data packets received:'+str(self.log_packet_count))
        self.log_packet_count = 0;
        logging.warning(text)
        print("Warning: %s" % text)


def print_incoming_text(self):
        """

        When starting the connection, print all the debug data until
        we get to a line with the end sequence '$$$'.

        """
        line = ''
        #Wait for device to send data
        time.sleep(1)

        if self.ser.inWaiting():
        line = ''
        c = ''
        #Look for end sequence $$$
        while '$$$' not in line:
        c = self.ser.read().decode('utf-8')
        line += c
        print(line);
        else:
        self.warn("No Message")

def openbci_id(self, serial):
        """

        When automatically detecting port, parse the serial return for the "OpenBCI" ID.
```

```
        """
        line = ''
        #Wait for device to send data
        time.sleep(2)

        if serial.inWaiting():
        line = ''
        c = ''
        #Look for end sequence $$$
        while '$$$' not in line:
        c = serial.read().decode('utf-8')
        line += c
        if "OpenBCI" in line:
        return True
        return False

def print_register_settings(self):
        self.ser.write(b'?')
        time.sleep(0.5)
        self.print_incoming_text();
#DEBBUGING: Prints individual incoming bytes
def print_bytes_in(self):
        if not self.streaming:
        self.ser.write(b'b')
        self.streaming = True
        while self.streaming:
        print(struct.unpack('B',self.ser.read())[0]);

        '''Incoming Packet Structure:
        Start Byte(1)|Sample ID(1)|Channel Data(24)|Aux Data(6)|End Byte(1)
        0xA0|0-255|8, 3-byte signed ints|3 2-byte signed ints|0xC0'''

def print_packets_in(self):
        while self.streaming:
        b = struct.unpack('B', self.ser.read())[0];

        if b == START_BYTE:
        self.attempt_reconnect = False
        if skipped_str:
        logging.debug('SKIPPED\n' + skipped_str + '\nSKIPPED')
        skipped_str = ''

        packet_str = "%03d"%(b) + '|';
```

```
        b = struct.unpack('B', self.ser.read())[0];
        packet_str = packet_str + "%03d"%(b) + '|';

        #data channels
        for i in range(24-1):
        b = struct.unpack('B', self.ser.read())[0];
        packet_str = packet_str + '.' + "%03d"%(b);

        b = struct.unpack('B', self.ser.read())[0];
        packet_str = packet_str + '.' + "%03d"%(b) + '|';

        #aux channels
        for i in range(6-1):
        b = struct.unpack('B', self.ser.read())[0];
        packet_str = packet_str + '.' + "%03d"%(b);

        b = struct.unpack('B', self.ser.read())[0];
        packet_str = packet_str + '.' + "%03d"%(b) + '|';

        #end byte
        b = struct.unpack('B', self.ser.read())[0];

        #Valid Packet
        if b == END_BYTE:
        packet_str = packet_str + '.' + "%03d"%(b) + '|VAL';
        print(packet_str)
        #logging.debug(packet_str)

        #Invalid Packet
        else:
        packet_str = packet_str + '.' + "%03d"%(b) + '|INV';
        #Reset
        self.attempt_reconnect = True


        else:
        print(b)
        if b == END_BYTE:
        skipped_str = skipped_str + '|END|'
        else:
        skipped_str = skipped_str + "%03d"%(b) + '.'

        if self.attempt_reconnect and (timeit.default_timer()-self.last_reconnect) >
self.reconnect_freq:
```

```
        self.last_reconnect = timeit.default_timer()
        self.warn('Reconnecting')
        self.reconnect()



def check_connection(self, interval = 2, max_packets_to_skip=10):
        #check number of dropped packages and establish connection problem if too large
        if self.packets_dropped > max_packets_to_skip:
        #if error, attempt to reconect
        self.reconnect()
        # check again again in 2 seconds
        threading.Timer(interval, self.check_connection).start()

def reconnect(self):
        self.packets_dropped = 0
        self.warn('Reconnecting')
        self.stop()
        time.sleep(0.5)
        self.ser.write(b'v')
        time.sleep(0.5)
        self.ser.write(b'b')
        time.sleep(0.5)
        self.streaming = True
        #self.attempt_reconnect = False



#Adds a filter at 60hz to cancel out ambient electrical noise
def enable_filters(self):
        self.ser.write(b'f')
        self.filtering_data = True;

def disable_filters(self):
        self.ser.write(b'g')
        self.filtering_data = False;

def test_signal(self, signal):
        if signal == 0:
        self.ser.write(b'0')
        self.warn("Connecting all pins to ground")
        elif signal == 1:
        self.ser.write(b'p')
        self.warn("Connecting all pins to Vcc")
        elif signal == 2:
```

```
        self.ser.write(b'-')
        self.warn("Connecting pins to low frequency 1x amp signal")
        elif signal == 3:
        self.ser.write(b'=')
        self.warn("Connecting pins to high frequency 1x amp signal")
        elif signal == 4:
        self.ser.write(b'[')
        self.warn("Connecting pins to low frequency 2x amp signal")
        elif signal == 5:
        self.ser.write(b']')
        self.warn("Connecting pins to high frequency 2x amp signal")
        else:
        self.warn("%s is not a known test signal. Valid signals go from 0-5" %(signal))

def set_channel(self, channel, toggle_position):
        #Commands to set toggle to on position
        if toggle_position == 1:
        if channel == 1:
        self.ser.write(b'!')
        if channel == 2:
        self.ser.write(b'@')
        if channel == 3:
        self.ser.write(b'#')
        if channel == 4:
        self.ser.write(b'$')
        if channel == 5:
        self.ser.write(b'%')
        if channel == 6:
        self.ser.write(b'^')
        if channel == 7:
        self.ser.write(b'&')
        if channel == 8:
        self.ser.write(b'*')
        if channel == 9 and self.daisy:
        self.ser.write(b'Q')
        if channel == 10 and self.daisy:
        self.ser.write(b'W')
        if channel == 11 and self.daisy:
        self.ser.write(b'E')
        if channel == 12 and self.daisy:
        self.ser.write(b'R')
        if channel == 13 and self.daisy:
        self.ser.write(b'T')
        if channel == 14 and self.daisy:
```

```
        self.ser.write(b'Y')
        if channel == 15 and self.daisy:
        self.ser.write(b'U')
        if channel == 16 and self.daisy:
        self.ser.write(b'I')
        #Commands to set toggle to off position
        elif toggle_position == 0:
        if channel == 1:
        self.ser.write(b'1')
        if channel == 2:
        self.ser.write(b'2')
        if channel == 3:
        self.ser.write(b'3')
        if channel == 4:
        self.ser.write(b'4')
        if channel == 5:
        self.ser.write(b'5')
        if channel == 6:
        self.ser.write(b'6')
        if channel == 7:
        self.ser.write(b'7')
        if channel == 8:
        self.ser.write(b'8')
        if channel == 9 and self.daisy:
        self.ser.write(b'q')
        if channel == 10 and self.daisy:
        self.ser.write(b'w')
        if channel == 11 and self.daisy:
        self.ser.write(b'e')
        if channel == 12 and self.daisy:
        self.ser.write(b'r')
        if channel == 13 and self.daisy:
        self.ser.write(b't')
        if channel == 14 and self.daisy:
        self.ser.write(b'y')
        if channel == 15 and self.daisy:
        self.ser.write(b'u')
        if channel == 16 and self.daisy:
        self.ser.write(b'i')

def find_port(self):
        # Finds the serial port names
        if sys.platform.startswith('win'):
        ports = ['COM%s' % (i+1) for i in range(256)]
```

```
        elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        ports = glob.glob('/dev/ttyUSB*')
        elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.usbserial*')
        else:
        raise EnvironmentError('Error finding ports on your operating system')
        openbci_port = ''
        for port in ports:
        try:
        s = serial.Serial(port= port, baudrate = self.baudrate, timeout=self.timeout)
        s.write(b'v')
        openbci_serial = self.openbci_id(s)
        s.close()
        if openbci_serial:
        openbci_port = port;
        except (OSError, serial.SerialException):
        pass
        if openbci_port == '':
        raise OSError('Cannot find OpenBCI port')
        else:
        return openbci_port

class OpenBCISample(object):
  """Object encapulsating a single sample from the OpenBCI board."""
  def __init__(self, packet_id, channel_data, aux_data):
        self.id = packet_id;
        self.channel_data = channel_data;
        self.aux_data = aux_data;
```

3.4.2. brain.py

```
import sys; sys.path.append('..') # help python find open_bci_v3.py relative to scripts folder
import open_bci_v3 as bci
import os
import logging
import time
from pygame.locals import *
import numpy
from collections import *
import math
```

```
import pandas as pd

port = '/dev/tty.usbserial-DQ0082W2'
baud = 115200
logging.basicConfig(filename="test.log",format='%(asctime)s - %(levelname)s :
%(message)s',level=logging.DEBUG)
logging.info('---------LOG START-------------')
board = bci.OpenBCIBoard(port=port, scaled_output=False, log=True)
print("Board Instantiated")
tInput = deque( [ ], maxlen = 250 )
e7_5Output = deque( [ ] )
e12_5Output = deque( [ ] )

def getsample( sample ):
        global e7_5Output
        global e12_5Output
        tInput.append( sample.channel_data[6] )
        frOutput = numpy.fft.fft( tInput )
        aOutput = abs(frOutput)
        pdOutput = pd.Series(aOutput)
        if pdOutput.get(33) != None:
        e7_5Output.append(pdOutput.get(33))
        if pdOutput.get(20) != None:
        e12_5Output.append(pdOutput.get(20))

def headto( ):
        board.start_streaming(getsample, 2.0)
        average7_5 = numpy.mean(e7_5Output)
        average12_5 = numpy.mean(e12_5Output)
        average = (average7_5 + average12_5)/2
        if 1000000 < average < 2000000:
        return "left"
        elif 2000000 <= average < 6000000:
        return "right"
        else:
        return "stay"
```

3.4.3 maze.py

```
import sys; sys.path.append('..')
import pygame
import open_bci_v3 as bci
import os
import logging
import time
from pygame.locals import *
import numpy
from collections import *
import math
import pandas as pd
import brain

pygame.init()

display_width = 800
display_height = 800

gameDisplay = pygame.display.set_mode((display_width,display_height))
pygame.display.set_caption('Maze Game by SSVEP')

white = (255,255,255)

exit = False
playerImg = pygame.image.load('player.bmp')
blockImg = pygame.image.load('block.bmp')

def player(x,y):
    gameDisplay.blit(playerImg, (x,y))

x = 250
y = 90
x_change = 0

def maze():
    M = 10
    N = 9
    maze = [ 1,1,1,1,1,1,1,1,1,1,
                    1,0,0,0,0,0,0,0,0,1,
                    1,1,1,1,1,1,1,1,0,1,
                    1,0,0,0,0,0,0,1,0,1,
```

```
                    1,0,1,1,1,1,0,1,0,1,
                    1,0,1,0,0,0,0,1,0,1,
                    1,0,1,1,1,1,1,1,0,1,
                    1,0,0,0,0,0,0,0,0,1,
                    1,1,1,1,1,1,1,1,1,1 ]
    bx = 0
    by = 0
    for i in range(0, M * N):
        if maze[ bx + (by * M) ] == 1:
            gameDisplay.blit(blockImg,( 200 + bx * 40 , 40 + by * 40 ))
        bx = bx + 1
        if bx == M:
            bx = 0
            by = by + 1

while not exit:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit = True

    if brain.headto() == 'right':
        x_change = 20
    elif brain.headto() == 'left':
        x_change = -20
    elif brain.headto() == 'stay':
        x_change = 0

    x += x_change

    gameDisplay.fill(white)
    player(x,y)
    maze()

    pygame.display.update()

pygame.quit()
quit()
```

3.4.4 Findings

Fig 1.4 Output of Maze game and user wearing OpenBCI Cap

Upon successfully running the code we can see that the maze pops up and the character moves in every 1.5 seconds interval. The data is recorded for 1.5 seconds, processed and then again data is recorded.

One limitation to note is that this project is very computationally intensive as handling so much real-time processing is difficult for normal gaming laptops.

Below is the screenshot of how the data captured looks. Upon running maze.py , you may not see the exact values sometimes but a message saying " waiting for buffer to clear", that i the default message that keeps appearing every time data processing occurs, if you see that message, but it keeps appearing in every 1.5 second interval, then there is nothing wrong, do not worry.

```
(base) PS C:\Users\Vansh Santdasani> cd SSVEP
(base) PS C:\Users\Vansh Santdasani\SSVEP> python open_bci_v3.py
(base) PS C:\Users\Vansh Santdasani\SSVEP> python maze_improved.py
pygame 2.5.2 (SDL 2.28.3, Python 3.11.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Connecting to V3 at port COM5
Serial established...
OpenBCI V3 8-16 channel
On Board ADS1299 Device ID: 0x3E
On Daisy ADS1299 Device ID: 0x3E
LIS3DH Device ID: 0x33
Firmware: v3.1.2
$$$
Board Instantiated
Captured data from channel 6: -3367832
Captured data from channel 6: -1166878
Captured data from channel 6: -6509434
Captured data from channel 6: -2300650
Captured data from channel 6: -6509089
Captured data from channel 6: -2304576
Captured data from channel 6: -6509205
Captured data from channel 6: -2322499
Captured data from channel 6: -6509170
Captured data from channel 6: -2292415
Captured data from channel 6: -6509151
Captured data from channel 6: -2324764
Captured data from channel 6: -6509397
Captured data from channel 6: -2300941
Captured data from channel 6: -6509214
Captured data from channel 6: -2304498
Captured data from channel 6: -6509373
Captured data from channel 6: -2322522
Captured data from channel 6: -6509250
Captured data from channel 6: -2292443
Captured data from channel 6: -6509279
Captured data from channel 6: -2324712
Captured data from channel 6: -6509458
Captured data from channel 6: -2301116
Captured data from channel 6: -6509353
Captured data from channel 6: -2304603
Captured data from channel 6: -6509554
Captured data from channel 6: -2323027
Captured data from channel 6: -6509275
```

Fig 1.5 Output of Readings

Verifying the movements

To check if the movements predicted are correct, we can verify it using the openBCI GUI. The GUI has a good accuracy rate and can show the exact graph of frequencies changing over time and a threshold being reached when the focus and relax are switched. The calibration and metrics we set in the code is based on what we observe in the GUI and modifications to the code metrics are made accordingly.

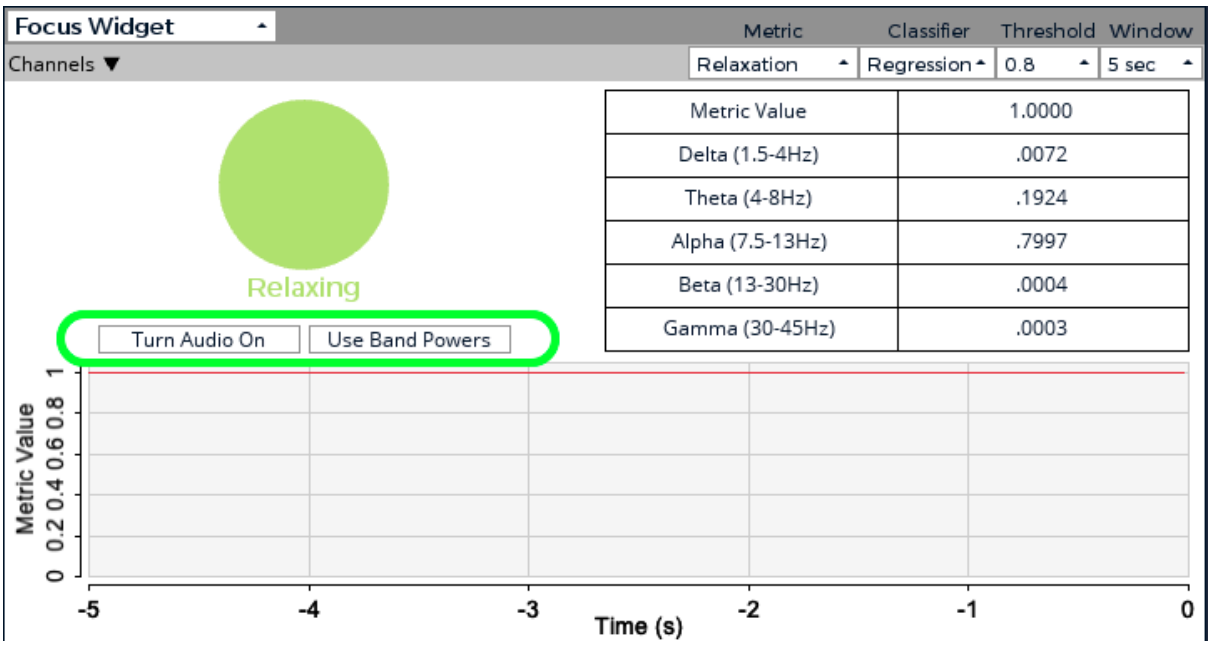

Fig 1.4 Focus Widget of OpenBCI GUI

5. Conclusion

In conclusion, this project represents a significant contribution in helping spread knowledge about EEG devices and Brain Computer Interface (BCI's). We strongly believe that by creating this interactive maze game, we have enlightened a curiosity in people's minds and that people would take this opportunity to create something extraordinary.

The primary objectives of this project were to demonstrate the basic abilities of BCI's and help people understand how it works through detailed steps, a well documented guide and providing the needed resources which we believe we have done.

Applications of BCI:

The potential applications of real-time EEG-based BCIs extend across diverse domains, promising to revolutionize various aspects of our lives:

Healthcare:

- **Neurorehabilitation:** Empowering individuals with motor impairments to regain control over limbs or prosthetics through the power of their thoughts.
- **Assistive Technologies:** Offering new avenues for communication and environmental control for individuals with disabilities, fostering greater independence and autonomy.

- **Mental Health and Pain Management:** Exploring potential applications in therapeutic interventions and pain management by monitoring and modulating brain activity.

Education and Entertainment:

- **Enhanced Learning:** Personalizing learning experiences, tracking cognitive engagement, and providing real-time feedback for optimized learning outcomes.
- **Immersive Entertainment:** Creating new frontiers in interactive gaming and entertainment, where users control characters or environments directly with their brainwaves.
- **Augmented Creativity:** Providing artists and musicians with novel tools for creative expression and interaction with digital art mediums.

Communication and Accessibility:

- **Alternative Communication:** Offering alternative communication channels for individuals with speech or language impairments.
- **Brain-Computer Control of Devices:** Enabling users to directly control smart homes, appliances, and assistive technologies using their thoughts.

These are just a glimpse into the vast potential of real-time EEG-based BCIs. By addressing the current challenges and fostering further innovation, we can unlock a future where the symphony of thought and technology seamlessly intertwines, shaping a world where our minds truly become the conductors of our reality.

References

[1] https://docs.openbci.com/Software/OpenBCISoftware/GUIDocs/

[2] https://openbci.com/community/getting-started-with-openbci/

[3] https://github.com/DarrenVawter/P300_BCI/tree/main?tab=readme-ov-file - To understand how the cap is connected

If you are interested in reading more about EEG based BCI and what all can be done with it, you are free to refer to the below references:

1. D. Zhang, L. Yao, K. Chen, S. Wang, X. Chang and Y. Liu, "Making Sense of Spatio-Temporal Preserving Representations for EEG-Based Human Intention Recognition," in IEEE Transactions on Cybernetics, vol. 50, no. 7, pp. 3033-3044, July 2020, doi: 10.1109/TCYB.2019.2905157.

2. Shahini, N.; Bahrami, Z.; Sheykhivand, S.; Marandi, S.; Danishvar, M.; Danishvar, S.; Roosta, Y. Automatically Identified EEG Signals of Movement Intention Based on CNN Network (End-To-End). Electronics 2022, 11, 3297. https://doi.org/10.3390/electronics11203297

3. Ou Bai, Peter Lin, Sherry Vorbach, Jiang Li, Steve Furlani, Mark Hallett, Exploration of computational methods for classification of movement intention during human voluntary movement from single trial EEG, Clinical Neurophysiology, Volume 118, Issue 12, 2007, Pages 2637-2655, ISSN 1388-2457, https://doi.org/10.1016/j.clinph.2007.08.025.

4. Shafiul Hasan, S.M., Siddiquee, M.R., Atri, R. et al. Prediction of gait intention from pre-movement EEG signals: a feasibility study. J NeuroEngineering Rehabil 17, 50 (2020). https://doi.org/10.1186/s12984-020-00675-5

5. Maryam Mahmoodi, Bahador Makkiabadi, Mehran Mahmoudi, Saeid Sanei, A new method for accurate detection of movement intention from single channel EEG for online BCI, Computer Methods and Programs in Biomedicine Update, Volume 1, 2021, 100027, ISSN 2666-9900, https://doi.org/10.1016/j.cmpbup.2021.100027.

6. Lin Yue, Hao Shen, Sen Wang, Robert Boots, Guodong Long, Weitong Chen, and Xiaowei Zhao. 2021. Exploring BCI Control in Smart Environments: Intention Recognition Via EEG Representation Enhancement Learning. ACM Trans. Knowl. Discov. Data 15, 5, Article 90 (October 2021), 20 pages. https://doi.org/10.1145/3450449

7. John Atkinson, Daniel Campos, Improving BCI-based emotion recognition by combining EEG feature selection and kernel classifiers, Expert Systems with Applications, Volume 47, 2016, Pages 35-41, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2015.10.049.

8. Torres, E.P.; Torres, E.A.; Hernández-Álvarez, M.; Yoo, S.G. EEG-Based BCI Emotion Recognition: A Survey. Sensors 2020, 20, 5083. https://doi.org/10.3390/s20185083

9. Schwartz AB, Cui XT, Weber DJ, Moran DW. Brain-controlled interfaces: movement restoration with neural prosthetics. Neuron. 2006 Oct 5;52(1):205-20. doi: 10.1016/j.neuron.2006.09.019. PMID: 17015237.

10. Pires, Gabriel & Nunes, Urbano & Castelo-Branco, Miguel. (2007). Single-Trial EEG Classification of Movement Related Potential. 569 - 574. 10.1109/ICORR.2007.4428482.

11. Tucker, M.R., Olivier, J., Pagel, A. et al. Control strategies for active lower extremity prosthetics and orthotics: a review. J NeuroEngineering Rehabil 12, 1 (2015). https://doi.org/10.1186/1743-0003-12-1