Investigation in Development and Applications of Elementary Electroencephalograms as Computer-Brain Interfaces

Rosnel Alejandro Leyva-Cortes

A project proposed for Alternate Study
Deerfield Academy, Deerfield, MA

1 Introduction

1.1 Overview

This paper is a technical overview of the GirasolEEG project. This project began in October of 2021 and is being reported on in its current state as of May 2022. This project is submitted to Deerfield Academy's office of Academic Affairs as part of the Directed Alternate Studies program.

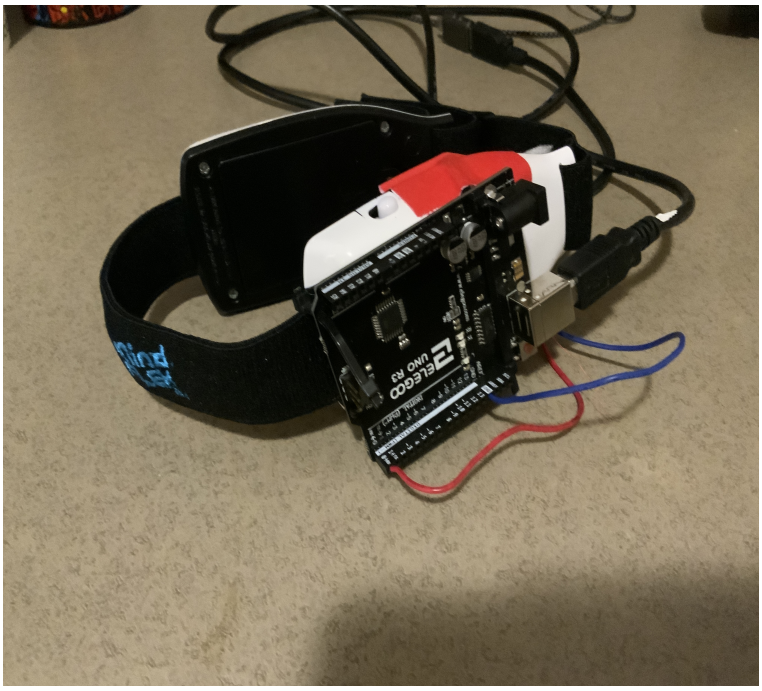
1.2 Purpose

The goal of this project is investigate the development of Electroencephalogram (EEG) devices for the purpose of using such devices as an alternative Human Input Device (HID). Currently, computers use a mouse as their primary mode of interfacing between humans and the machine. The latency between the communication of the brain, to the fingers, and then to the mouse, and finally being interpreted by the computer affects the workflow of an environment that prioritizes high throughput of information. Additionally, a neural interface would benefit handicapped individuals who find themselves unable to interact with traditional HIDs. This paper will start with the development of such a headset, and then delve into the development of software that was used to allow for the control of the computer cursor using a neural interface. Inspiration for this project came from a paper in the Columbia University Undergraduate Science Journal. [1]

2 Materials

This section outlines materials that were used for the creation of this headset. All materials were bought during September 2021, availability of these materials may vary.

2.1 Electroencephalogram Headset



Pictured above is the Mindflex EEG Headset created by Neurosky. Neurosky develops many EEG devices that serve as toys marketed at a young audience. This model comes from a set designed to control motors in an external toy. This headset measures electrical activity of the brain from a range of 1Hz - 50Hz. The headset features three electrodes, two null references electrodes that attach to the earlobes, and the principal electrode that attaches to the base of the forehead. In healthcare, these headsets are often invasive, attaching to the base of the scalp, and are often used to diagnose epilepsy.

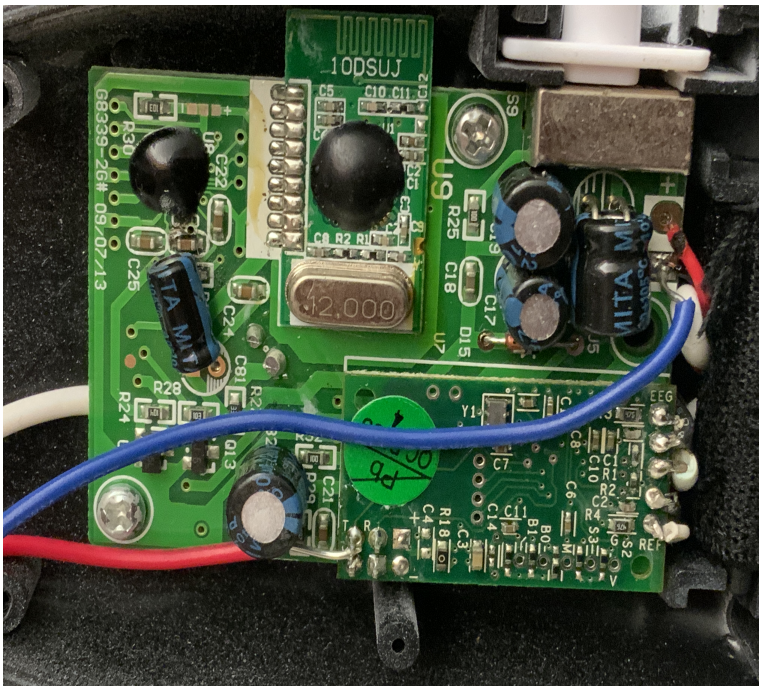
2.2 Micro-Controller

The micro-controller used in this project Arduino Uno R3 manufactured by Eleggo. The image in the previous section features said micro-controller. This board is used as a "middle-man" that interprets data from the headset and relays it to the computer.

2.3 A Note on Setup

The EEG contains a development board with an output pin and a pin for ground. In order to catch packets of communication from the EEG, the micro-controller must have its receiving serial pin (Rx) connected to the output of the headset, and the ground of the micro-controller linked to the ground of the EEG. Pictured

below is the wiring of the headset where the red wire denotes the Rx wire, and blue denotes ground.



2.4 Software

This project does not deal with parsing serial communication between the EEG and the Arduino micro-controller. This problem was solved and its implementation made open-source by Eric Mika's project on the matter. [2]

This project uses said library to read communication from the EEG, as we will only focus on using this input data to control a target device.

Languages used in this project range from C++ for programming the micro-controller board, to Python for the interpretation of data and manipulation of cursor elements on a target device. All software was run on a MacBook Pro (13-inch, M1, 2020)

3 Implementation

The following sections go over the development of key methods and functions that drive the software of this project.

3.1 Arduino Code

Code for the Arduino micro-controller is written in C++. The Arduino requires Mika's library [2]. We use the instantiated "Brain" module to update readings from the EEG and send that data from the Arduino to an active serial port. For this project, the same serial port used for programming, will also be used to read the serial output from the Arduino. Using the 9600 baud reading speed, the board will continuously update its reading every second. At this point, we take readings of signal quality, attention, and low gamma that are all separated by a comma delimiter. Signal quality ensures that the readings we take in from the headset are accurate. Attention is a pre-processed value computed by the onboard chip of the EEG and only produces output when signal quality is perfect. The low gamma wavelength readings are affected by multi-sensory input (such as talking, moving, or listening to music). The endbit is unreadable as a "byte" object and is only used to denote the end of serial communication.

```
#include <Brain.h>

// Set up the brain parser, pass it the hardware serial object you want
    to listen on.
Brain brain(Serial);

void setup() {
    // Start the hardware serial.
    Serial.begin(9600);
}

void loop() {
    // Expect packets about once per second.
    if (brain.update()) {

      //For general use
        String signalQuality = String(brain.readSignalQuality());
        String attention = String(brain.readAttention());
        String gamma = String(brain.readLowGamma());
        String endbit = String(0);
        Serial.println(signalQuality + "," + attention + "," + gamma +
            "," + endbit);
    }
}
```

3.2 Python Code

Using the code from the previous section, a computer will be able to take in the information being processed by the Arduino in the format of a String in the following form:

$$"Signal, Attention, LowGamma, EndBit" \quad (1)$$

The following methods from the serialParser object parse the string using commas as a delimiter, then it instantiates each item as a byte object in an array monikered myByteList. Thus, to retrieve signal quality, or attention values, it is trivial to have the computer wait for a desired duration (in seconds) and use the properly indexed value in myByteList that corresponds with the desired value.

```
#returns signal quality value
#a null object is returned when the signal quality drops
#a byte like object is returned if successful
#works as of May 4th !
def getSignalQuality(duration):



    startTime = time.time()
    while(time.time() < startTime + duration):
        read_serial = ser.readline()



    myByteList = read_serial.split(b",")
    #returns byte object for signal quality
    return myByteList[0]

#returns attention value
#for a duration in seconds(int)
#works as of May 4th
def getAttention(duration, read_serial=None):

    startTime = time.time()
    #this is to ensure that we have a healthy signal
    # 0 is full connection, 200 is no connection to headset
    # recall that these are byte like objects as well

    while (time.time() < startTime + duration and
        getSignalQuality(duration)== b'0') :
        read_serial = ser.readline()

    myByteList = read_serial.split(b",")
    return myByteList[1]

#Returns the low gamma (31-40Hz) power value, associated with
    multi-sensory processing.
# I've found it more associated with physical movement
def getGamma(duration, read_serial=None):
    startTime = time.time()
    # this is to ensure that we have a healthy signal
```

```
    # 0 is full connection, 200 is no connection to headset
    # recall that these are byte like objects as well

    #remember to check signal quality as in other methods
    while (time.time() < startTime + duration and
        getSignalQuality(duration) == b'0'):
        read_serial = ser.readline()

    myByteList = read_serial.split(b",")
    return myByteList[2]
```

The main runner contains the methods that take data from the serialParser object and interpret it into actions on the computer cursor. I used the PyAutoGui library [3] to control the cursor. The library is written in objective-c and features a suite of control methods that range from keyboard to touchscreen automation.

The algorithms for cursor control are as follows:

Movement in the X direction on the screen is assigned to fluctuations in attention values. If attention values surpass a pre-determined constant, and attention values are strictly increasing, then the cursor will move in the positive X direction (conventionally, it will move to the right). If values are less than the constant, and are strictly decreasing, it will move in the negative X direction. The same logic follows for movement in the Y direction using the data from the "gamma" bits. The finalized implementation is within the standardOp() method.

I've also included a method that ignores the signal quality of the data being interpreted in the scenario that movement in one axis must be prioritized over quality of the readings.

```

#this moves the cursor in the y direction depending
#on if it passes a threshold
def mouseYmovement(duration):
    if(serialParser.getGamma(duration) >= gamma_const):
        print('move up')
        moveUp(duration)
    if (serialParser.getGamma(duration) <= gamma_const):
        print('move down')
        moveDown(duration)

#this moves the cursor in the x direction depending
#on if it passes a threshold
def mouseXmovement(duration):
    if(serialParser.getAttention(duration) > attn_Const):
        print('move right')
        moveRight(duration)
    if (serialParser.getAttention(duration) < attn_Const):
        print('move left')
```

```
        moveLeft(duration)

#this moves the cursor up
def moveUp(duration):
    initVal = serialParser.getGamma(duration)
    startTime = time.time()
    # as long as the current value is constantly decreasing
    while (time.time() < startTime + duration):
        while (serialParser.getGamma(duration) >= initVal):
            pyautogui.move(0, -mouse_y_duration, mouse_duration)

#this moves the cursor down
def moveDown(duration):
    initVal = serialParser.getGamma(duration)
    startTime = time.time()
    # as long as the current value is constantly decreasing
    while (time.time() < startTime + duration):
        while (serialParser.getGamma(duration) <= initVal):
            pyautogui.move(0, mouse_y_duration, mouse_duration)

#this will move the cursor to the left
#works as of May 6th
def moveLeft(duration):
    initVal = serialParser.getAttention(duration)
    startTime = time.time()
    #as long as the current value is constantly decreasing
    while(time.time() < startTime + duration ):
        while(serialParser.getAttention(duration) <= initVal):
            pyautogui.move(-mouse_x_duration,0,mouse_duration)

#this will move the cursor to the right
#works as of may 6th
def moveRight(duration):
    initVal = serialParser.getAttention(duration)
    startTime = time.time()
    #as long as the current value is constantly increasing
    while(time.time() < startTime + duration):
        while (serialParser.getAttention(duration) >= initVal):
            pyautogui.move(mouse_x_duration,0,mouse_duration)

#This method, finalized on the week of May 13th
def standardOp(duration):
    #I'm trying to make a failsafe to leave this script in case it goes
        bad
    try:

        while(True):
            mouseXmovement(duration)
            mouseYmovement(duration)
    except KeyboardInterrupt:
```

```
        print("You have interuptted Brain Interface Device")
        print("Restart Script")

#NOTE: This is a fallback option
# in an ideal world, use the standardOp method only
# I made this method to serve as a workaround b/c
# the signal quality of my headset had deteriorated
def workAroundOp(duration):
    # I'm trying to make a failsafe to leave this script in case it goes
        bad
    try:

        while (True):
            failsafeXmovement(duration)

    except KeyboardInterrupt:
        print("You have interuptted Brain Interface Device")
        print("Restart Script")

def failsafeXmovement(duration):
    if(serialParser.getGamma(duration) >= gamma_const):
        print('move right')
        pyautogui.move(mouse_x_duration,0,mouse_duration)
    if (serialParser.getGamma(duration) <= gamma_const):
        print('move left')
        pyautogui.move(-mouse_x_duration,0,mouse_duration)
```

Within the main runner class, the standardOp method is intended run. While running, the cursor can still be manipulated through traditional HID input, however additional control is given to the headset that supersedes control by a trackpad and/or mouse. Implemented also, is a keyboardInterrupt catch statement, which allows the code to be terminated anytime by using the machine specific interrupt code. For Macs it is traditionally $Ctrl + C$.

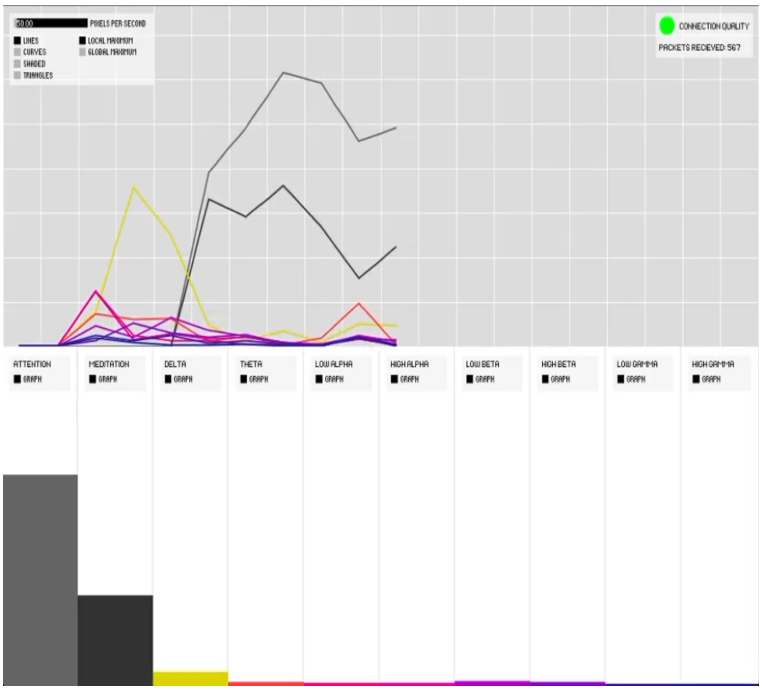
4 Testing

At this point, data is able to be taken in, however the malleability is still unknown. In this project I conducted brief testing on myself to determine what signals from the EEG would be most convenient for controlling a mouse cursor. I used the BrainGrapher repository from Eric Mika [4] that generates a visualization of the serial data that is being interpreted by the Arduino.

4.1 Methodology and Data

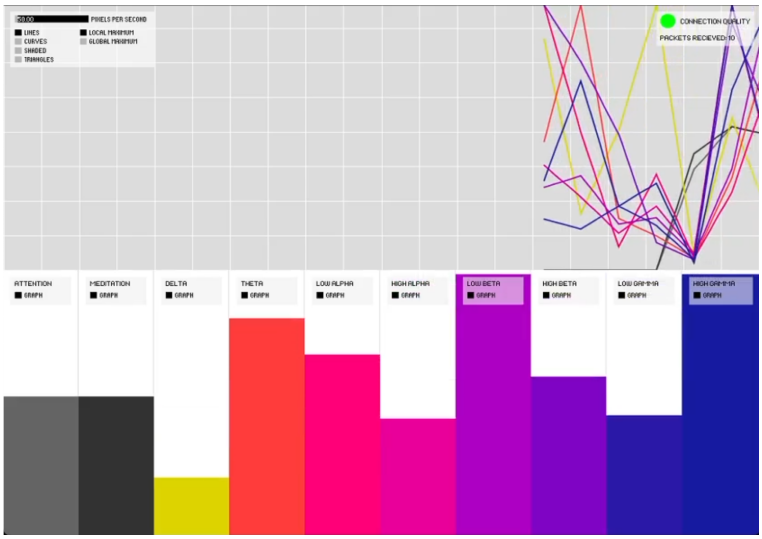
I chose attention and gamma for my code, therefore I attempted a few activities and monitored the fluctuation in these values from the Braingrapher to see if there was any way of manipulating them.

To test attention values I fixated on the color of the table that my laptop rested on, not allowing myself to think of anything else. Using functions built into the BrainGrapher, the average attention values appeared as follows after 5 trials:



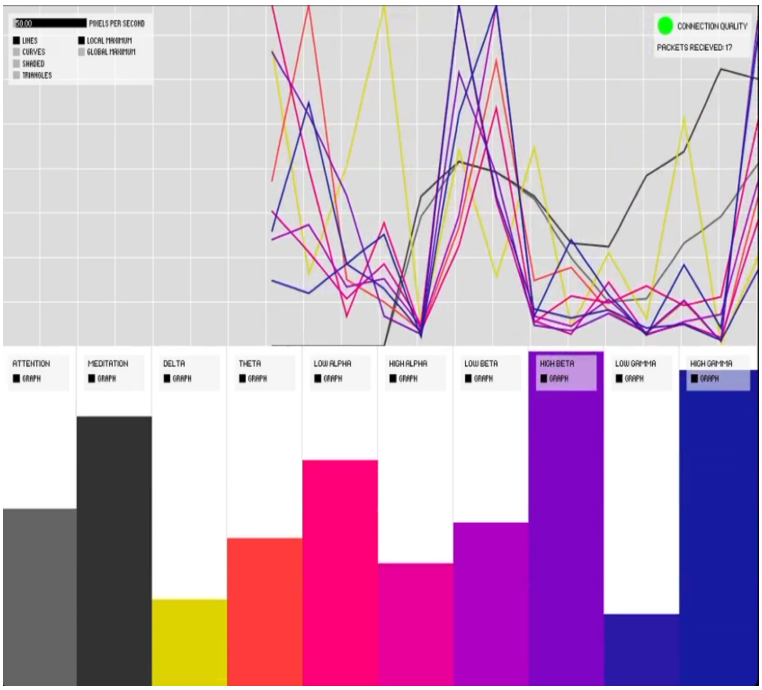
Attention values visibly spiked, with all other values being effectively negligible.

To test gamma values I attempted multiple multi-sensory stimulating activities. I conducted 5 trials of doing "jazz-hands" for 2 minutes and found the following fluctation in values:



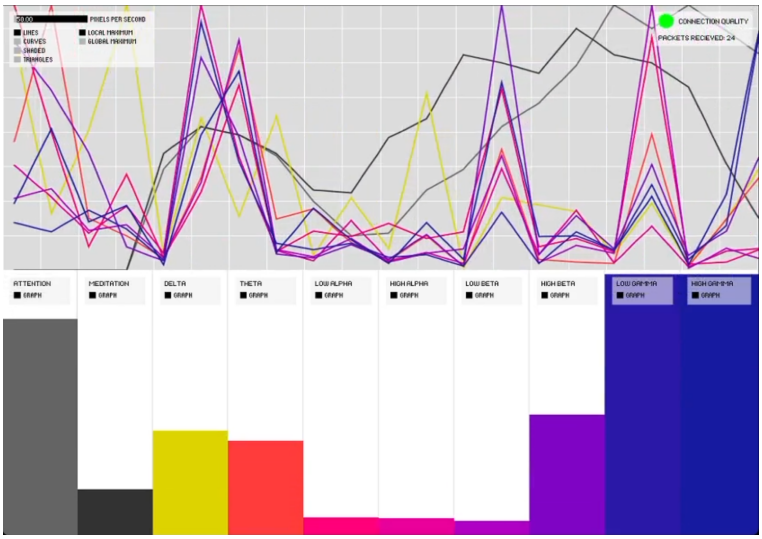
While high-gamma was stimulated the most, I found that low-gamma and other values seemed to contain chaotic behavior that was not reproducible after each trial.

I then attempted to watch comedy sketches, using laughter and audio/visual input, as the catalyst activity for low-gamma values. I found the following fluctuations after watching 5, 5 minute comedy sketches:



This proved to be unreliable as well, since it appeared that virtually all values spiked when undertaking this activity.

Finally, I conducted 5, 5 minute conversations between myself and acquaintances whilst having the headset mounted to my head. I found the following fluctuations:



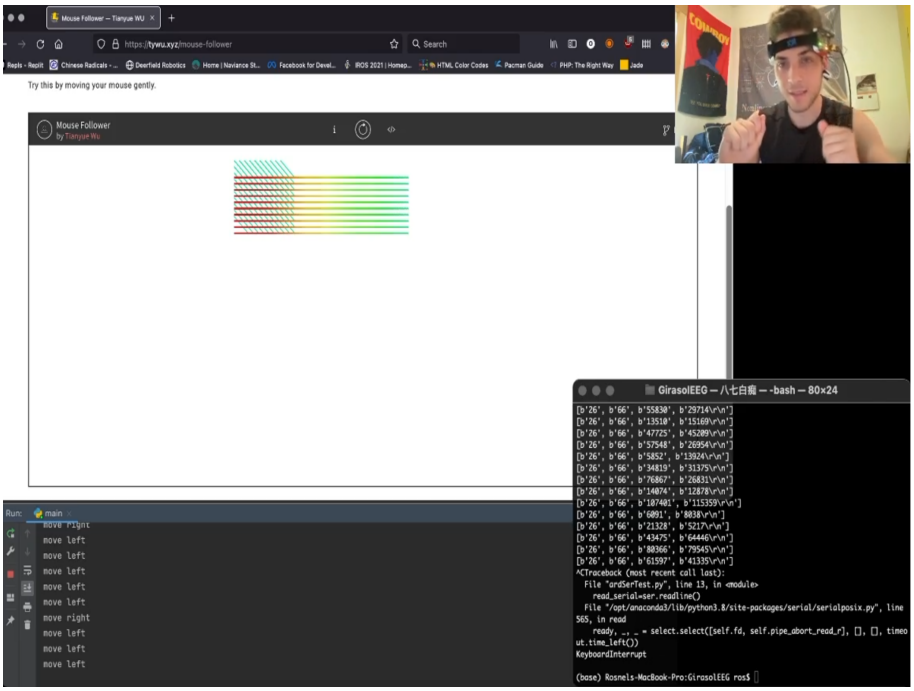
4.2 Reflection

There was much more consistent manipulation of low-gamma values in particular. This cemented the justification for using low-gamma and attention values as input for manipulating the cursor on a computer.

5 Demonstration

To demonstrate the functionality of final product, I utilized software from Tian Yu Wu [5] to highlight the position of the cursor as a color gradient. I had to adapt the functions to only move in the X direction because the signal quality deteriorated during this phase of testing. From an implementation standpoint, I used the fail-safe method but in a scenario where signal quality isn't variable the standardOp method would be used.

A snapshot of this demonstration can be found here:



[6]

Further documentation of processes and the demonstration can be found at the cited resources for this image.

6 Conclusion

Overall, the current implementation, while functional, is not as "smooth" as I would like it to be. I've had to introduce a delay that samples values from the EEG after a given duration, which makes the sampling rate longer than just using traditional HIDs. It was still amazing to see a computer controlled by the manipulation of my thoughts, however, I'm sure that with further development the latency of the device can be decreased. Developments from the company, Neuralink, have shown the Brain-Computer interfaces introduce a more accessible future for individuals that are unable to use traditional HIDs, and in a bio-medical context can even be used to address illnesses of the brain induced by abnormal electrical activity. The future of these devices will likely be pushed forward by methods that use invasive electrodes that pierce the scalp or epidermal layer of the user, however, being able to exhibit some level of control on elementary devices such as the Mindflex EEG headset show a promising future for the development of such interfaces.

References

[1] V. Ogunniyi, D. Abugaber, I. Finestrat, A. Luque, and K. Morgan-Short, "Predicting second language proficiency with resting-state brain rhythms," *Columbia Undergraduate Science Journal*, vol. 15, p. 39–54, 2021.

[2] E. Mika, "Kitschpatrol/brain: Arduino library for reading neurosky eeg brainwave data. (tested with the mindflex and force trainer toys.)."

[3] "Pyautogui's documentation."

[4] E. Mika, "Kitschpatrol/braingrapher: Processing-based visualizer for neurosky eeg brainwave data output from the arduino brain library.."

[5] W. Tian Yu, "Mouse follower."

[6] R. A. Leyva-Cortes, "Eeg video documentation."