Controlling a Video Game with an EEG

Thomas Woodside

5/15/19

In this project, an electroencephalogram (EEG) was built to record activity of brain waves and electrical impulses in the head. The EEG was used to control a game of Flappy Bird through electrical signals from a player's blinks. A series of amplifiers and filters were built to convert the very small electrical signals into usable output, and software was written that allows control of the game. Although the device was not as accurate at measuring brain signals as initially hoped, it proved capable of controlling the game through muscle signals. In addition, there is some evidence that the device can measure a user's concentration.

Motivation and History

The idea for the project came from the ASR website, and was also influenced by an example from an instructables page [1] which proved that such a project was feasible with consumer electronics.

Systems for controlling video games have evolved immensely over the years. Spacewar, arguably the first video game, was controlled using only a series of switches, which was cumbersome because it required players to remember what each switch did. Later, buttons and joysticks were used to control video games, and today, many controllers use a combination of the two [2]. The Xbox 360, developed in 2009, allowed users to control games using gestures [3]. Today, tools like the Microsoft Hololens allow gameplaying to be conducted in augmented reality based on the motions of the player [4]. The late physicist Stephen Hawking, unable to speak and afflicted by ALS, could control his computer by twitching his cheek [5].

Luigi Galvani, an Italian physician, in 1791 became the first to recognize that nerve impulses are electrical, by stimulating animal tissue with electrical probes. He viewed the brain as the center of these electrical impulses [6]. In 1803, his brother Giovanni Aldini was the first to apply this theory to human tissue, when he used electricity to move the limbs of an executed criminal in the streets of London [7].

In 1875, Richard Caton placed electrodes on opposite sides of the brain and found that currents flowed through, increasing during sleep, though he didn't measure anything more than the magnitude of the currents. The variations of the currents could not be explained by any other factor aside from electrical activity in the brain. The first basic EEG was created in 1924 by Hans Berger, a German physician, who built on Caton's results to identify changes in brain activity associated with attention and brain injury [8]. Wilder Penfield, a Canadian physician, was the first to identify the function of various parts of the brain through electrical stimulation, using this knowledge to treat epilepsy [9] (the author's great-grandfather was one of Penfield's patients).

In 1988, an EEG was used to control a robot: the robot continued in a straight line when the subject had their eyes closed, and stopped when they opened their eyes [10]. In September 2018, researchers at the University of Washington and Carnegie Mellon University created an interface that allowed three users to communicate using only their brains and collaborate to play a game of tetris. The players were able to control the game by looking at one of two LEDs, one blinking at 15hz and the other at 17hz. The difference in the power observed in the corresponding brain wave frequencies from a standard EEG was used to determine which LED the player was focusing on [11]. Similar controls are likely to be useful for this project.

In the future, it is likely that more and more interfaces will be controlled through the brain via some version of EEG. Although controls are still rudimentary, they are likely to improve rapidly, creating a new and engaging way to play video games. In addition, the technology could one day be applied to help people with conditions that do not allow them to use ordinary controls, such as those with ALS or those who are otherwise paralyzed. The EEG could be an improvement over devices like Stephen Hawking's, which are much slower than ordinary typing.

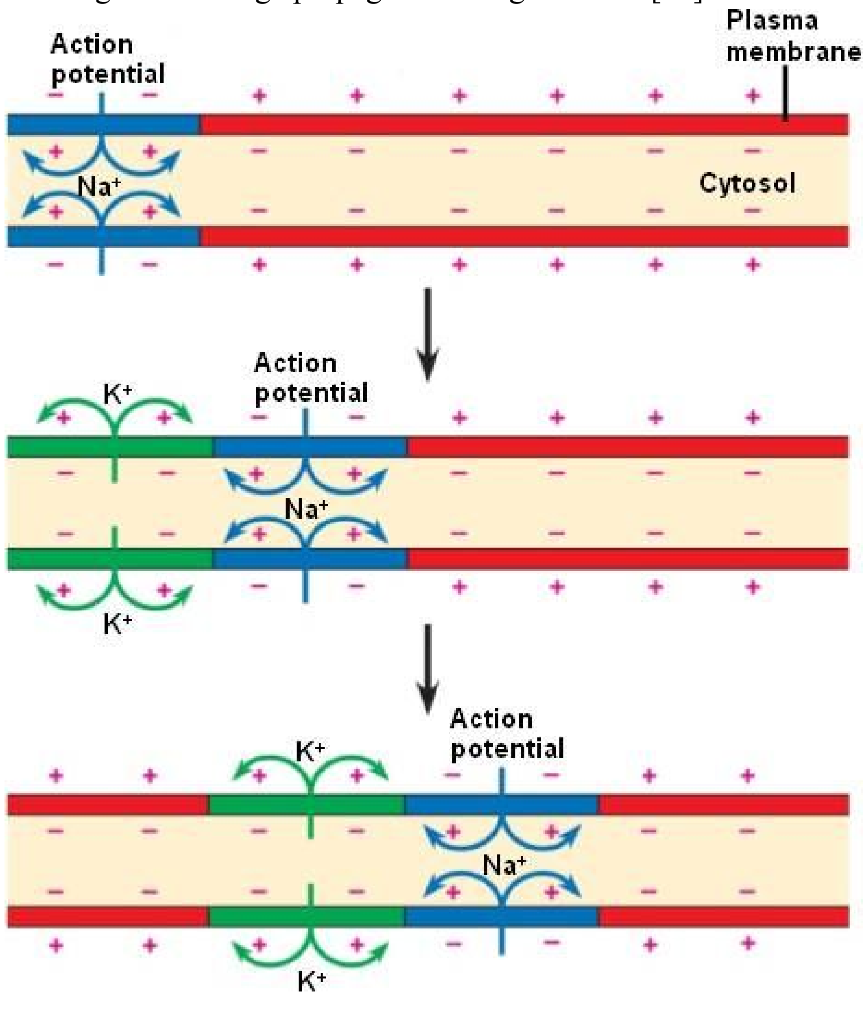
The Neuroscience

The brain works through electrical impulses carried between neurons, which can be measured from the outside of the brain.

The inside of neurons are kept negatively charged through several mechanisms. First, pumps found in the membranes of the neurons pump 3 NA+ out of the cell for every 2 K+ they pump into the cell. In addition, passive K+ channels allow K+ to flow back out of the cell, creating still more negative charge inside the cell. [12]

The negative charge inside the cell can be leveraged to create an action potential, which allows charge to flow from one neuron to another through axons. Axons operate mainly through voltage-activated gated ion channels, which move charge alone their membranes. First, NA+ ions on the inside of the cell membrane cause pumps further down the axon to activate and pump in NA+. K+ then flows out of the membrane, equalizing the charge in the first region of the axon. The positive charge differential propagates down the axon until it reaches another neuron (see Figure 1). Such signals are involved in all perception and cognition in the brain. As such, any brain activity creates electrical signals and voltage differences that can be measured to some extent using specialized equipment. [12]

Figure 1: Charge propagation along the axon [13]



Electroencephalography, or EEG, is one way to measure electrical signals in the brain. In particular, it measures neural oscillations, which are relatively large differences in voltage across the head. Rather than being the signals from any one neuron, neural oscillations are the composite of the signal from many neurons. There is some debate as to whether neural oscillations play an integral role in brain function or are simply the result of electric potentials in the brain, but they can certainly illuminate information about the state of the brain. EEGs can be used to determine sleep states, concentration, and some visual signals, although they can't yet decipher more complex emotions and thoughts. [14]

EEG can also detect electrical potential in muscles along the scalp, such as those activated when blinking or moving, and it is thus important to control for those factors when trying to measure signals only in the brain. [15]

The occipital lobe, located at the back of the head, is responsible for visual processing in the brain. Changes in visual stimuli can cause neural oscillations in the occipital lobe, which, being near the skull, is also a good location for an EEG. Thus, for this application, one electrode is placed on the occipital lobe and another on the frontal lobe, a configuration which maximizes the EEG effectiveness as it is reading the oscillations across the entire skull. [16]

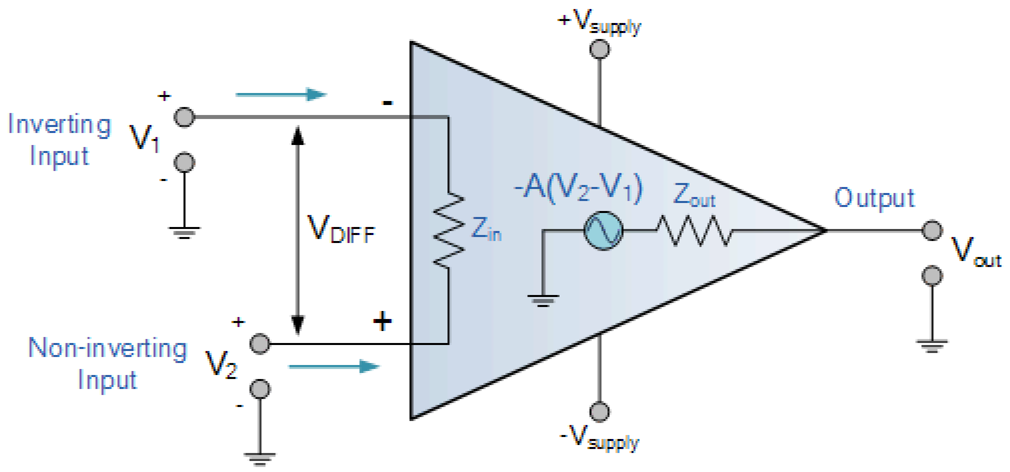
Brain waves consist of four frequency types, ranging from 1 to 30hz. Delta and theta waves, which range from 1 to 8 hz, are typically most active in sleep, so are not useful for this application. Alpha waves, which range from 8hz to 13hz, and beta waves, which range from 13hz to 30hz, are active when awake, during relaxed states and alert states, respectively, and are thus most useful in this application. [17]

The Circuitry

To control a video game, electrodes placed on the head detect electrical signals from brain waves and muscle impulses as voltage differences between disparate parts of the head. Brain wave frequencies are amplified and filtered to eventually be sent to a microcontroller.

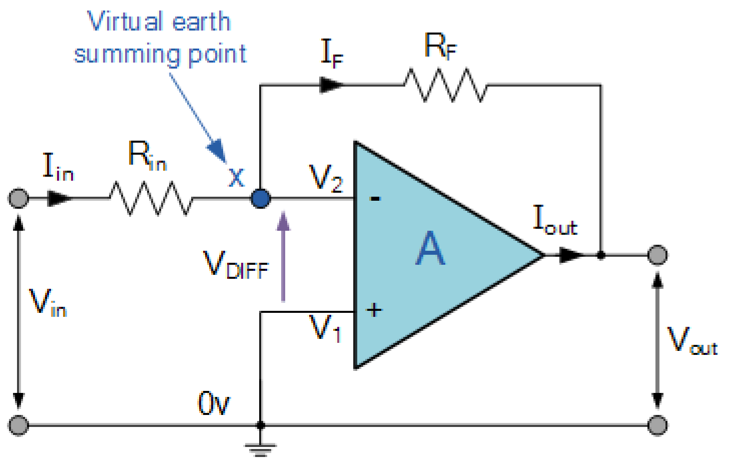
Voltage amplifiers and filters are commonly constructed using operational amplifiers (op amps). Op amps have two inputs and an output, and in their most basic form serve as difference amplifiers with infinite gain. An ideal op amp can be seen in Figure 2, where Z_{in} is infinite and Z_{out} is zero. In their "open loop" configuration, if the non-inverting input is at a higher voltage than the inverting input, the output will supply the lowest voltage allowed by the power supply. If the inverting input has a higher voltage, the output will supply a high voltage.

Figure 2: An ideal op amp [18]



Op amps are usually used in a closed-loop configuration, where the output is connected to one of the inputs. This causes the op amp to only produce enough voltage to equalize the inputs, at which point the op amp reaches a state of equilibrium. In the closed-loop configuration, resistors can be used to achieve a particular gain, rather than simply the default infinite gain of the op amp in the open-loop configuration. In Figure 3, the op amp resistor values can be set to invert the input voltage and multiply it by a particular gain, equivalent to R_f/R_{in} in the diagram.

Figure 3: An inverting op amp [19]



More complicated designs can be used to amplify the difference between the two inputs, which is useful if neither are at ground, as can be seen in Figure 4. If R1 and R3 are set equal to each other and R4 and R2 are also set equal, then the amplifier can amplify the difference between the two inputs successfully with gain equal to R2/R1 [20].

Brain waves typically have very low amplitudes, ranging from 5 to 200μV [17]. In order to properly measure the brain waves, their amplitudes must be amplified to have a maximum amplitude of around 5V, which is the full voltage range of the microcontroller. The ideal gain would thus be around 25,000.

As discussed previously, a simple amplifier can be created using a single operational amplifier (see Figure 4). However, this design will also amplify voltage that is common to both inputs (common-mode signal), which is likely to be far greater than the very small differences created by brain waves. A better design consists of three operational amplifiers (see Figure 5).

Since amplification is set via R_g, and not through the initial op amps themselves, the common-mode signal is not amplified, while the difference is amplified through the final op amp.

Figure 4: A Single Op Amp Amplifier [21]

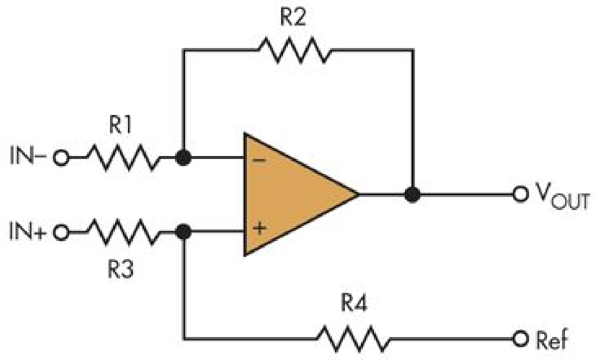
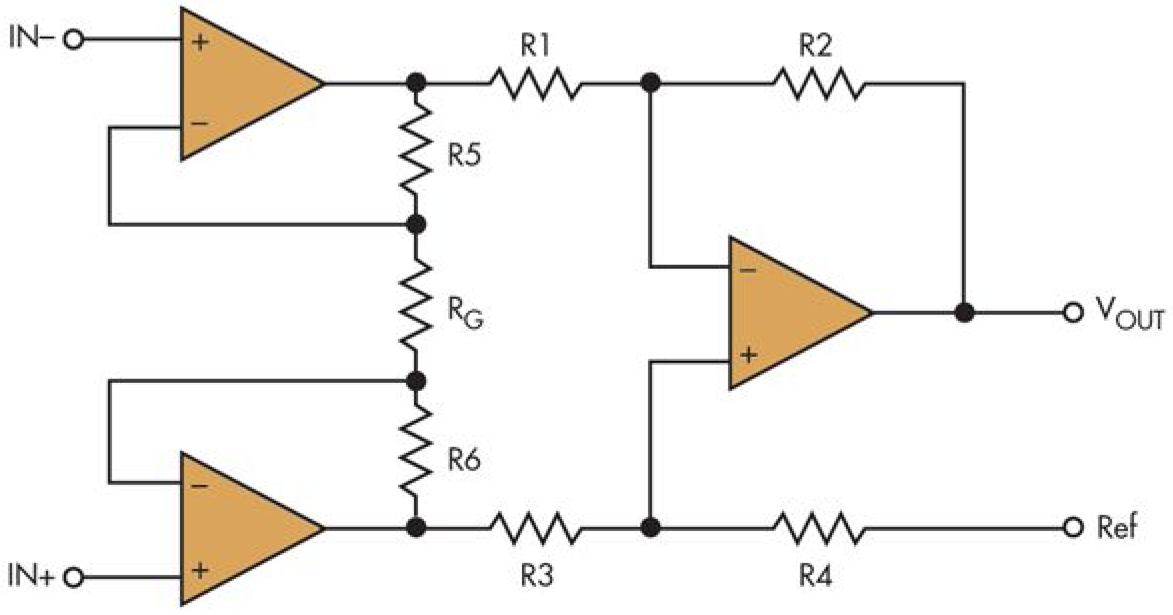


Figure 5: A Three Op Amp Instrumentation Amplifier [22]



The common-mode rejection ratio (CMR), a measure of how much the amplifier rejects the common-mode inputs, is equal to $20log[(100*(1+R_1/R_2))/R_t]$ where R_t is the mismatch between the ratios of the resistor pairs as a fraction: $|1-(R_1/R_2)/(R_3/R_4)|$. Stated differently, small differences in the ratios of the resistor pairs can make a large difference in the CMR [22].

Because the three op amp approach that can be built in the lab is liable to having low CMR, and thus reduced performance, an instrumentation amplifier was purchased. The amplifier has precision-cut resistors that are temperature resistant, maximizing the CMR [23].

The instrumentation amplifier is controlled by a single resistor, and gain is determined by the equation $49.4k\Omega/R_G + 1$. The maximum gain of the amplifier is 10,000, meaning that additional amplification will be needed later [23]. A potentiometer is used to more easily fine tune the exact gain of the circuit (Figure 6.1).

Figure 6.1: First Half of Circuit Diagram

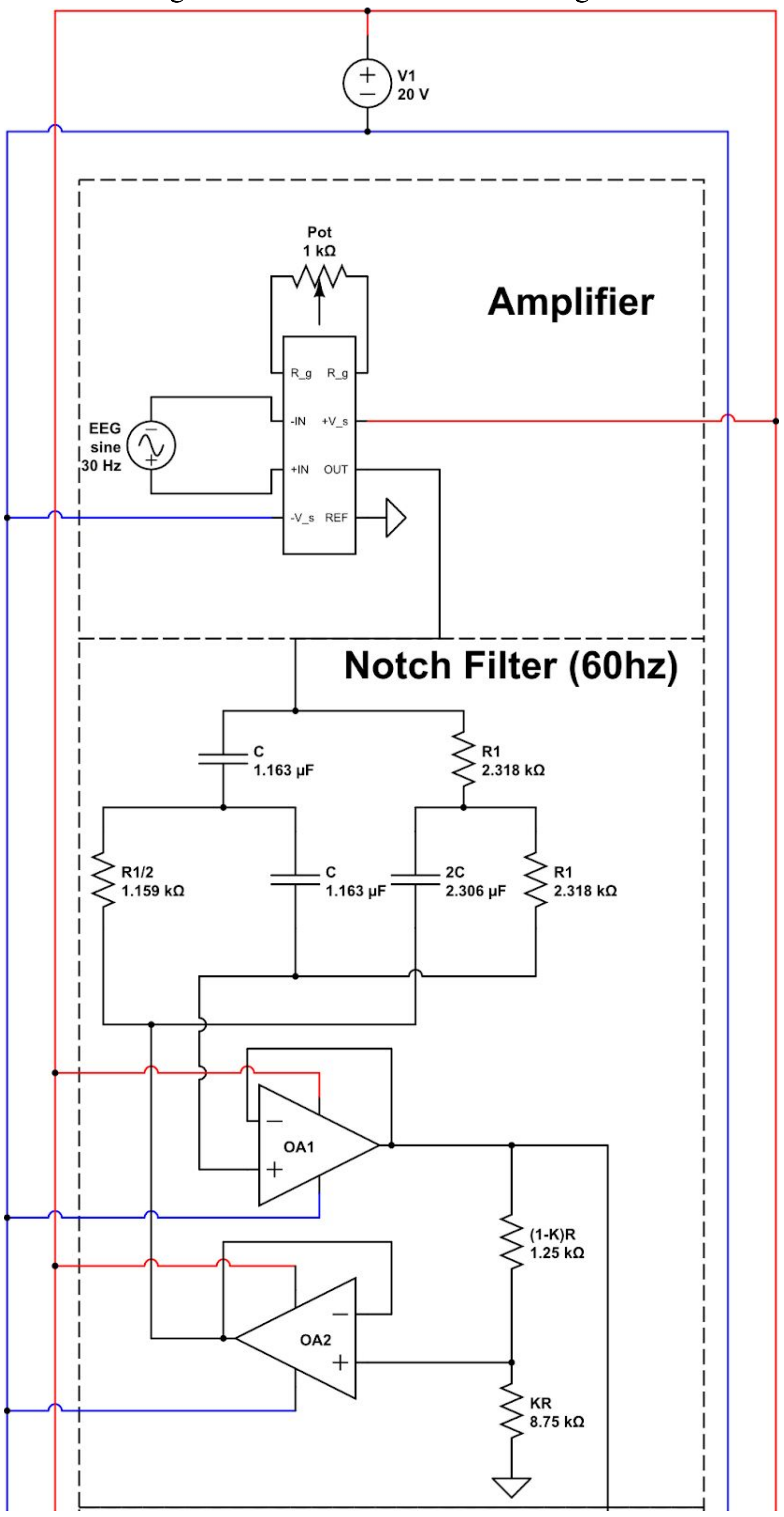


Figure 6.2: Second half of circuit diagram

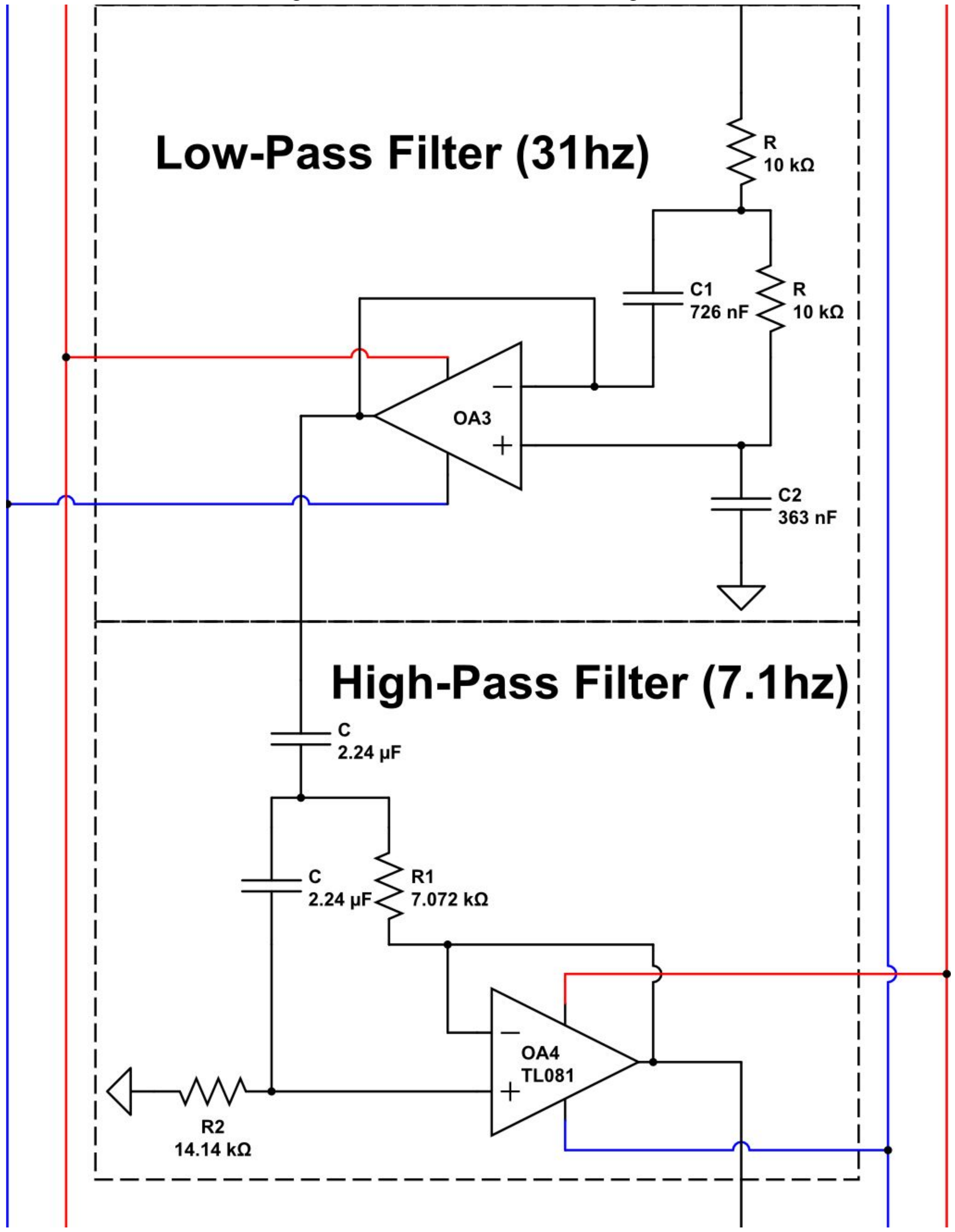
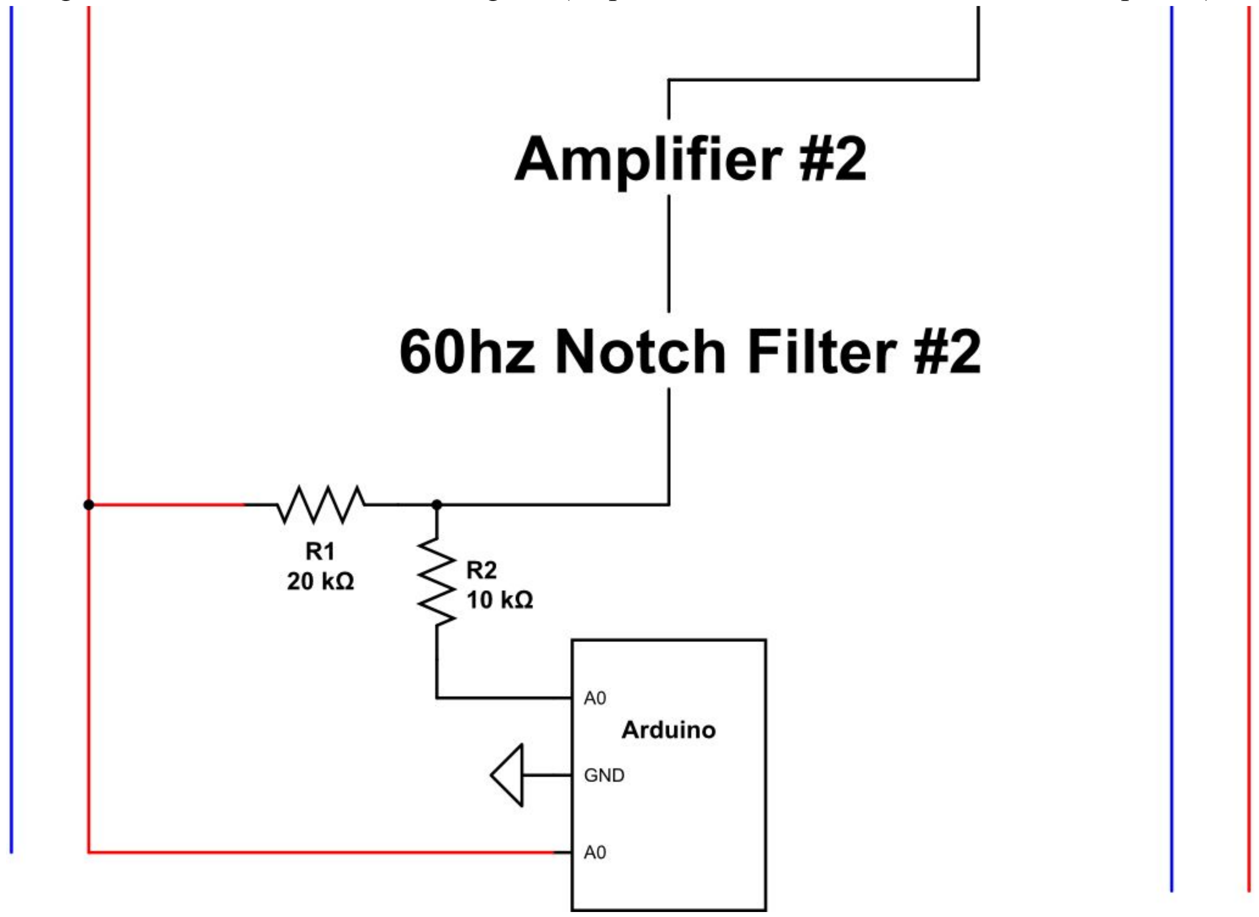


Figure 6.3: Final third of circuit diagram (amplifier and notch filter from first third repeated)



To properly focus on only signals from alpha and beta waves, signals of other frequencies that are the result of interference must be filtered out. All filters below are modeled off of active filters detailed in a book in the lab [24].

In addition to their use as amplifiers, op amps are also useful to create active filters that are capable of filtering out particular frequencies, which is very helpful for this application. Specifically, such filters use the varying reactance of capacitors as given by the formula $X_C = 1/(2\pi fC)$ where X_C is the reactance f is the frequency of the signal, and C is the capacitance of the capacitor. Because the reactance of capacitors vary by the frequency of the signal passing through, capacitors are useful for removing particular frequencies from a signal.

The first filter necessary for the EEG must remove unwanted interference from power lines. Known as the "mains hum," the interference is concentrated at 60hz, and could potentially be much greater than the actual signal from the electrodes [25]. To properly filter out the mains hum, a 60hz notch filter was designed (see Figure 6.1). A 30Hz -3-dB bandwidth was chosen to maximize attenuation; since only signals below approximately 30hz are necessary, the range of the notch filter with that bandwidth (starting at around 30hz and ending at around 90hz) should not affect signal. The 30hz bandwidth requires the filter to have a Q, or ratio between notch frequency and -3-dB range, of 2. A Twin-T passive notch filter, as shown in Figure 7, has a constant Q of 0.25 that cannot be easily varied based on experimental results. For this reason, an active notch filter was used, as shown in Figure 7.1. The calculations used to determine the final resistor and capacitor values are shown in Table 1.

Figure 7: A Twin-T passive notch filter [24]

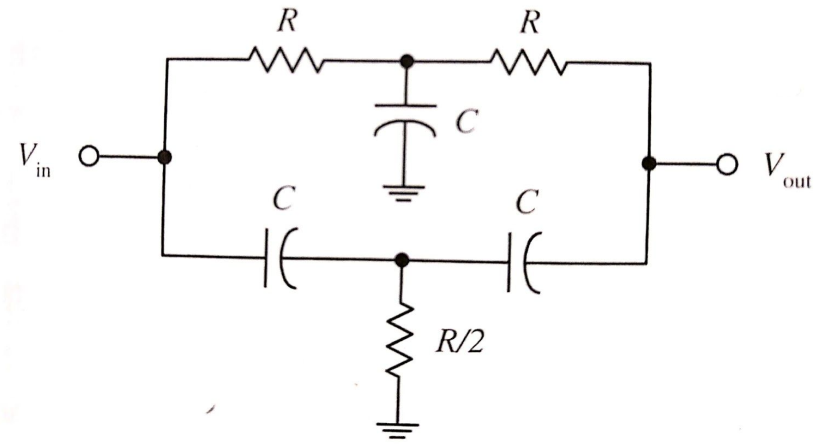


Table 1: Notch Filter Calculations

Name	**Value**
Notch Frequency (A)	60Hz
-3-dB bandwidth (B)	30Hz
Q = A/B	2
R (arbitrary)	10KΩ
C (arbitrary)	**1.163µF**
R_1 = 1/(2πAC)	**2,281Ω (actual: 2,318Ω)**
R_1/2	**1,141Ω (actual: 1,159Ω)**
K = (4Q -1)/4Q	0.875
(1-K)R	**1.25KΩ**
KR	**8.75KΩ**

When the notch filter was implemented using resistors from the lab, it mainly filtered noise centered around 62hz, not 60hz, likely due to a decrease in resistance caused by temperature. Resistors were then gradually added in series to decrease the center of the notch (the frequency of the notch has an inverse relationship with the resistance). The measured value of R_1 (2.318KΩ) was thus about 1.6% higher than the calculated theoretical value (2.281KΩ).

After the notch filter, a low-pass filter (see Figure 3.2) was designed to remove frequencies that are higher than the highest-frequency brain waves, as they are likely to be noise. An active low-pass filter was designed with a -3dB frequency of 30hz. In this case, it is not as critical to have an active filter, as passive filters will have relatively quick attenuation, but an active filter was chosen because the use of operational amplifiers ensures that the filter is isolated from other electronic components, and reduces the risk of inductors picking up additional interference. The calculations used to determine the resistor and capacitor values for the filter are shown in Table 2.

Table 2: Low-Pass Filter Calculations

Name	**Value**
C_1 (from table)	1.414
C_2 (from table)	0.7071
R (arbitrary)	**10KΩ**
f_{3dB}	30hz
$C_{1\text{-actual}} = C_1 / (R*2\pi*f_{3dB})$	**750nF**
$C_{2\text{-actual}} = C_2 / (R*2\pi*f_{3dB})$	**375nF**

Next, a high-pass filter (Figure 3.2) was designed for a 7hz -3dB frequency to filter out signal that is lower frequency than brain waves. An active filter was again used, for the same reasons as for the low-pass filter. The calculations used to determine the resistor and capacitor values for the filter are shown in Table 3.

Table 3: High-Pass Filter Calculations

Name	**Value**
C_1 (from table)	1.414
C_2 (from table)	0.7071
R (arbitrary)	10KΩ
R_{table}	1Ω
f_{3dB}	7.1hz
$C_{transformed} = 1/R_{table}$ (in F)	1F
$R_1 = 1/C_1$ (in Ω)	0.7072Ω
$R_2 = 1/C_2$ (in Ω)	1.414Ω
$C_{actual} = C_{transformed} / (R*2\pi*f_{3dB})$	**2.24μF**
$R_{1\text{-actual}} = R*R_1$	**7,072Ω**
$R_{2\text{-actual}} = R*R_1$	**14.14KΩ**

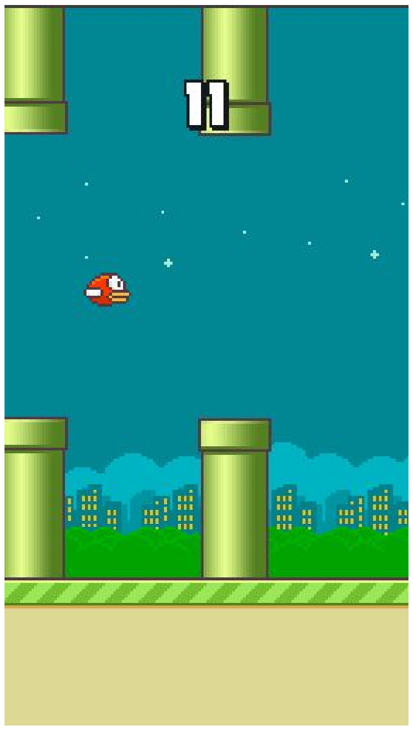
Two of the circuits already mentioned are repeated to provide maximal signal. Another amplifier is used to amplify the signals so that they can be fully within the Arduino's range, and another 60hz notch filter is placed at the end to remove additional noise, including noise built up as the signal travels through the circuit.

The output from the circuit was then fed into an Arduino microcontroller at an analog input pin. The oscillating output was almost always within a volt of the overall circuit's ground, which itself was around 10 volts from the circuit's positive and negative voltage. For this reason, if the Arduino was grounded at the same ground, and powered with the high voltage rail, it would only be able to detect about half of the signal (Arduino cannot read voltages below its ground). To remedy this, a voltage divider was built, with a 20KΩ resistor going from the output to the high voltage, and a 10KΩ resistor going from the output to the Arduino input pin. With this setup, the average output voltage was around 3.3 volts (10/(10+20)) instead of 0 volts, which allowed the Arduino to read the entirety of the output.

In order to read at the highest frequencies possible, the Arduino was programmed to only read from the analog input pin and send that reading to a computer over serial. Otherwise, no processing was done on the Arduino. This reduced time on the Arduino's small processor and ensured the maximum number of readings were sent to the computer. The Arduino could read in and relay the signal at a frequency of 191hz.

The Flappy Bird game was essentially the one built and open sourced by Sourabh Verma, with a few modifications [26]. First, in traditional Flappy Bird, the pipes are very close together, making it difficult to play even by tapping. To make the game easier to play using blinking, the distance between the pipes was doubled. See Figure 8 for a screenshot of the modified game. In addition, a calibration procedure was added to the game, which will be discussed in more detail later in the paper.

Figure 8: A screenshot of the Flappy Bird game.

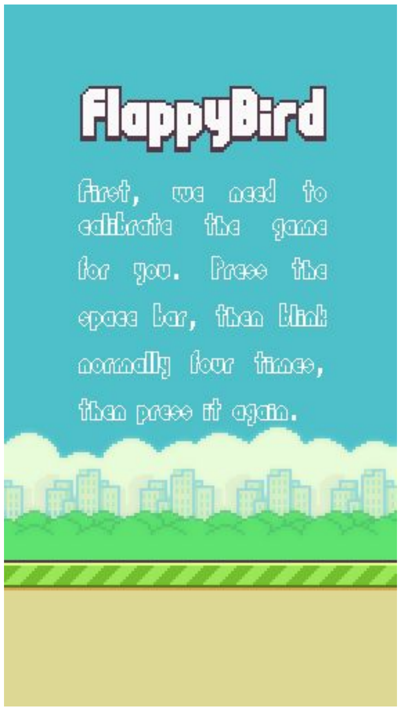


On the computer, a Python script was used to read in the serial output from the Arduino using the Python serial library. The output was read repeatedly in a loop and processed immediately. Initially, the program was designed to recognize a blink when the signals from the Arduino were above a threshold value (for instance, 700) for five consecutive readings, and then went back under that level. However, five consecutive readings was often not enough to ensure that the user was really blinking and that the observation wasn't due to noise. In response, the program was modified to require eight consecutive high readings.

Even more importantly, the threshold value often needed to be adjusted depending on the person playing, the amount of electrode gel applied, and the configurations of the potentiometers controlling the amplifier. This adjustment was difficult while the user was playing the game, and meant that the game took a very long time to set up. To remedy this, a calibration stage was built in to the program.

When the game is first started, the user sees a screen with instructions to press the spacebar, blink four times, and press the spacebar again (see Figures 9 and 10). The screen can also be activated after the game is over by pressing "c". In the background, the program records all measurements while the user is blinking. When the user is finished blinking, the program processes the data starting from a low initial threshold value of 600. With a value that low, the program will identify many more blinks than actually occur. The program then progressively raises the threshold value until it reaches a point where it identifies exactly four blinks. It records that number, and continues to raise the threshold until only 3 blinks or fewer are identified. At this point, the program has identified the highest and lowest thresholds capable of detecting the four blinks. The program then sets the threshold for game playing to the average of the two. A flow chart of this process can be found in Figure 11, and the full code can be found in the Appendix.

Figure 9: The first calibration screen shown to the user.


FlappyBird

First, we need to calibrate the game for you. Press the space bar, then blink normally four times, then press it again.

Figure 10: The screen shown while the user is calibrating.

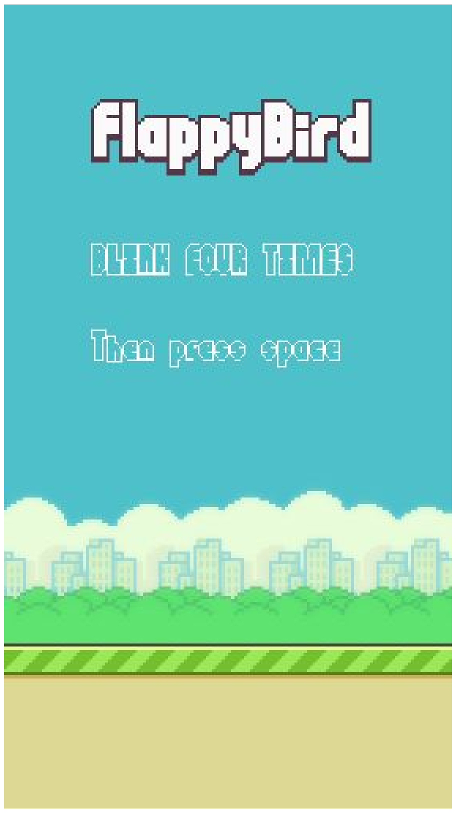
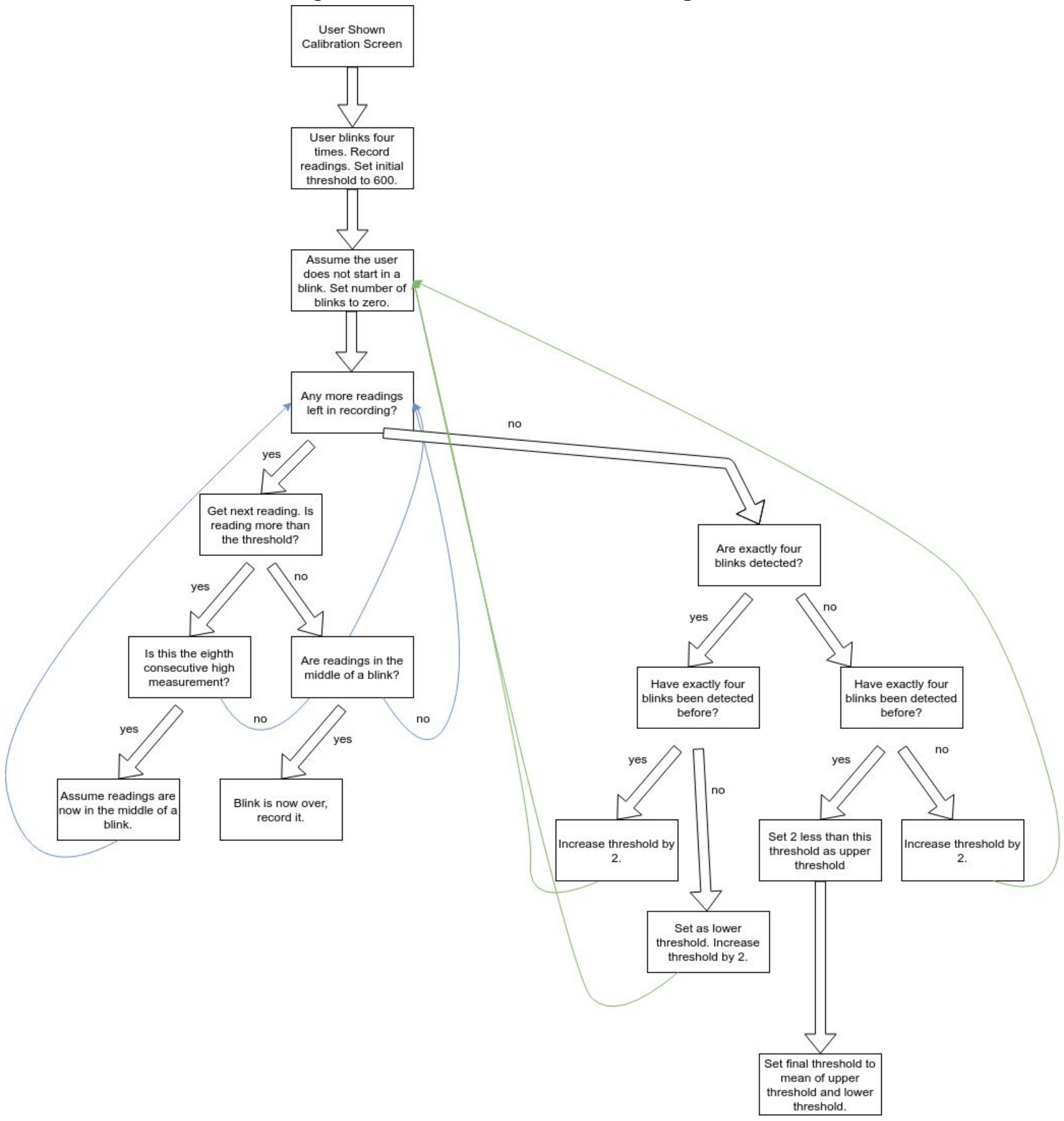


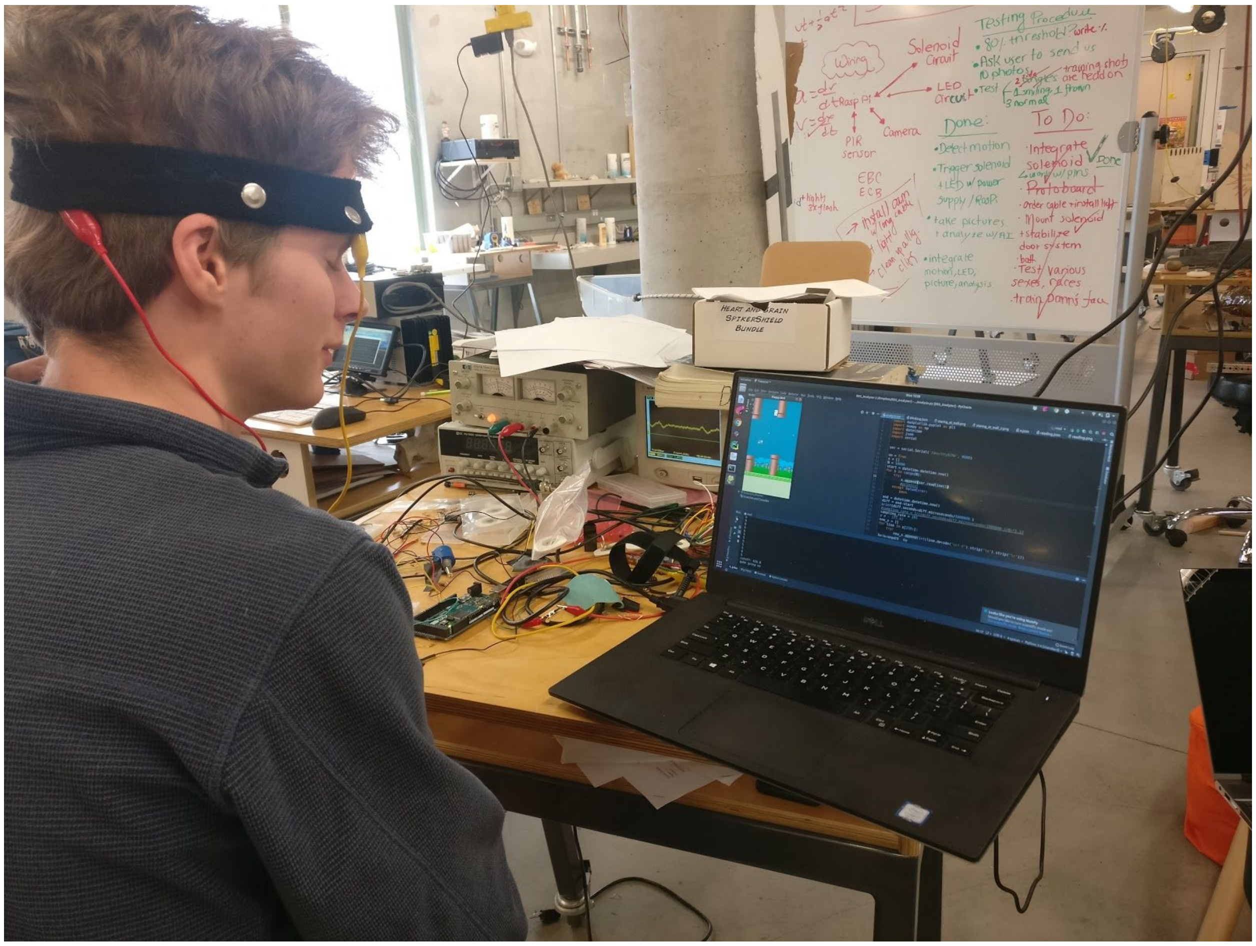
Figure 11: Flow chart of calibration process.


User Shown Calibration Screen
User blinks four times. Record readings. Set initial threshold to 600.
Assume the user does not start in a blink. Set number of blinks to zero.
Any more readings left in recording?
yes
no
Get next reading. Is reading more than the threshold?
yes
no
Is this the eighth consecutive high measurement?
Are readings in the middle of a blink?
yes
no
yes
no
Assume readings are now in the middle of a blink.
Blink is now over, record it.
Are exactly four blinks detected?
yes
no
Have exactly four blinks been detected before?
Have exactly four blinks been detected before?
yes
no
yes
no
Increase threshold by 2.
Set as lower threshold. Increase threshold by 2.
Set 2 less than this threshold as upper threshold
Increase threshold by 2.
Set final threshold to mean of upper threshold and lower threshold.

After calibration, the game begins with the determined threshold value. Whenever a blink is detected, the Python pyautogui library is used to send the signal (a spacebar input) to the main game. A blink can also be used to restart the game after it is over.

A photo of the full setup of the project is shown in Figure 12.

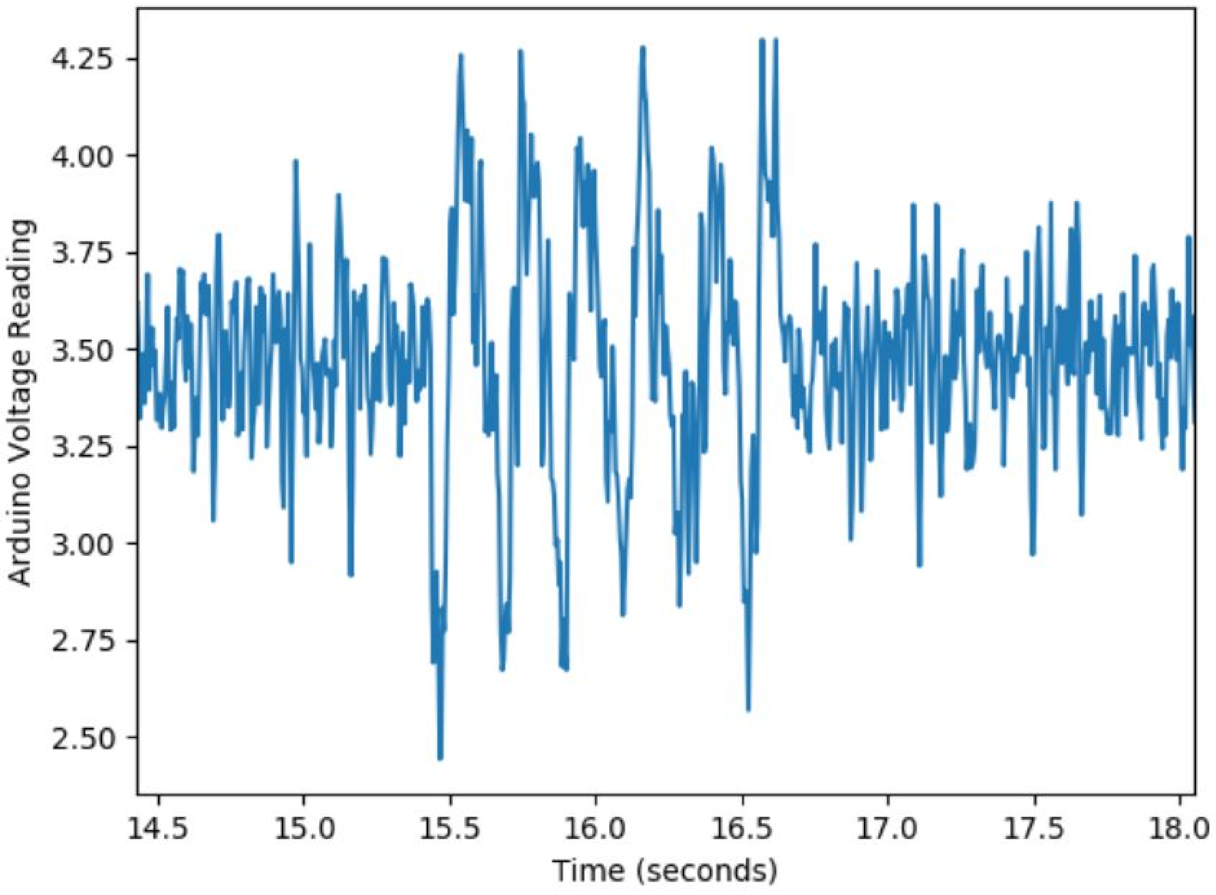
Figure 12: Project setup



Results

The game was tested on a total of 10 people. Blink detection worked very well on seven of them, rarely missing blinks or identifying blinks that were not there. On two of them, it was much less reliable, usually because it failed to identify any blinks. With one person, no blinking signal was ever detected at all, even on oscilloscope readings. A graph illustrating the readings produced by a series of blinks is shown in Figure 13.

Figure 13: The signal pattern of a series of five blinks. Note the peaks at each blink.



One major impediment to perfect measurements was that there were only around 9 total electrodes, which meant that they all needed to be reused. The electrodes clearly functioned less well after copious reuse, and sometimes fell apart. Another problem was using the device on people with long or very thick hair, as it was difficult for the electrode in the back of the head to make any contact with the skin.

At first, when the device was tested outside, the signal had large amounts of interference. It was determined that this was due to the fact that the users were not properly grounded when on the fake grass. This problem was resolved by instructing users to hold a grounded wire with their hand.

As stated earlier in the paper, the original goal of the project was to use brain waves to control the game. In this implementation, using only blinking, it mainly read electrical impulses in the muscles on the scalp. However, the system did have some ability to read brain waves. To analyze this, a test subject (the author) was fitted with electrodes while performing three tasks: reading, staring at the wall, and closing their eyes. All tests were conducted in a seated position while looking directly forward, and lasted approximately 50 seconds (10,000 readings). Power graphs were created from the data using a Fast Fourier Transform, and relative power was calculated for both alpha and beta waves by taking the mean power in their spectra and subtracting the mean power of all frequencies.

Results of the experiment are shown in Table 4. As expected, alpha waves tended to be much more powerful when the user had their eyes closed. Figures 14 and 15 show a visible difference in the power graphs. This effect was not observed in the second trial with eyes closed, for unknown reasons. It is possible that the user was being distracted by voices or other thoughts while completing the second experiment with eyes closed. Reading also appeared to produce a weak increase in beta waves compared to the other experiments, as anticipated, but results were not strong enough for a definite conclusion.

However, contrary to expectations, higher levels of alpha waves were not observed when the test subject was staring at the wall than when they were reading, and beta waves were not lower when the subject's eyes were closed than when they were staring at the wall.

Table 4: Results of Experiments with various brain states.

Activity	Relative Power of Alpha Waves	Relative Power of Beta Waves
Eyes closed	17.91, 9.60, 15.22 (14.24)	7.51, 8.87, 8.33 (8.24)
Staring At Wall	10.98, 9.86 (10.42)	8.74, 6.76 (7.75)
Reading	11.65, 11.11 (11.38)	8.95, 9.43 (9.19)

Figure 14: A spectral density graph of brain waves with eyes open. Note the 60hz noise.

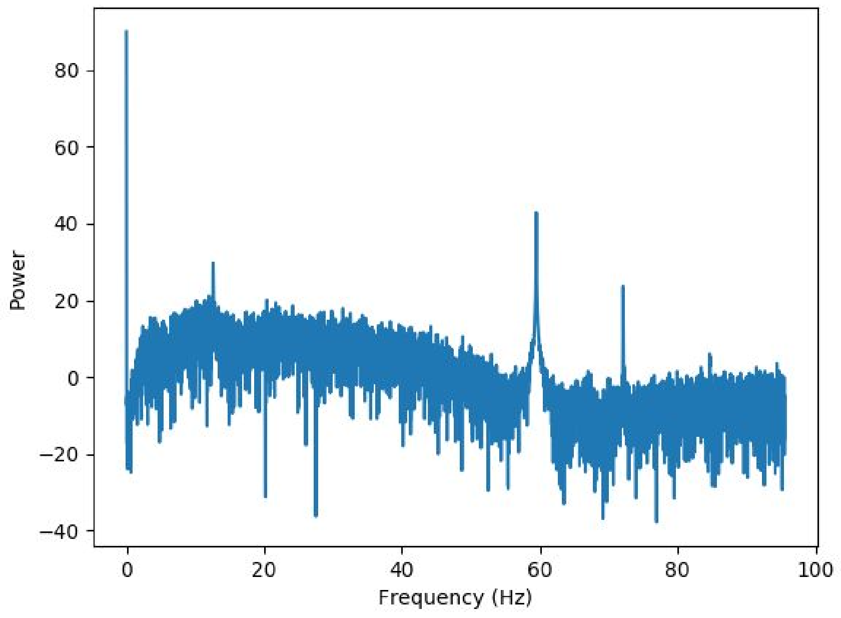
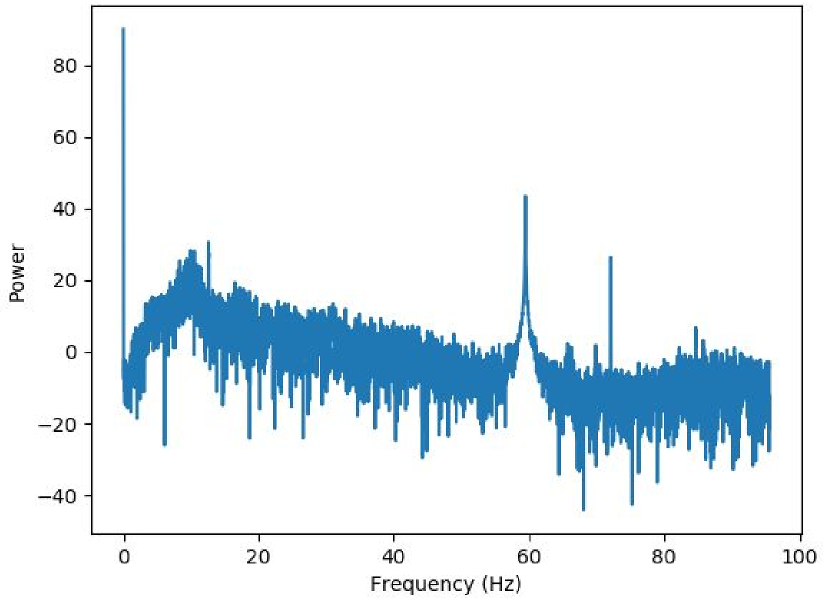


Figure 14: A spectral density graph of brain waves with eyes closed. Note the high levels of alpha waves around 7-13hz.



Conclusion

The original goal of the project, to use brain waves to control a video game, turned out to be too ambitious for completion within the timeframe, despite some encouraging results in experiments. However, a working prototype was produced that could successfully use electrical impulses produced by blinking to control a game, and the prototype worked on most people that it was tested on.

The experiments on brain wave activity were plagued by a number of systematic errors. First, it is very difficult to perfectly control concentration in an experiment, as anybody who has ever gotten distracted will know. Second, despite the filters, there was still a good deal of noise in the input, which increased the variability of the data. Lastly, the electrodes used were far from state of the art and did not always produce consistent readings.

The initial experiments do suggest that the circuit could be able to identify a user's concentration, at least on a rudimentary level, if improvements were made to the filters and electrodes. The ability to detect concentration could be useful in a wide variety of contexts, including work, school, and even meditation. Even as it is, the project presents a new and interesting way to control a video game which most players had never experienced, and demonstrated that it is possible to read brain waves using very rudimentary equipment.

Next Steps

The project would likely be dramatically improved by purchasing higher quality components: precision resistors and capacitors, better electrodes, and better connection wires. In addition, interference would likely be reduced further by moving the circuit to a protoboard, constructing a Faraday cage to house the components, and finding a way to connect the computer directly to the oscilloscope instead of the Arduino for higher quality output.

With these improvements, the project could likely be expanded to test the user's concentration and allow more interesting applications, such as a device that could warn users when they were getting unfocused.

Even without any improvements to the hardware, the software could be configured to allow the user to type by blinking, for instance using Morse code. Such a device could allow people unable to move, such as those with ALS, to be able to type.

Acknowledgements

I would like to acknowledge Dr. Dann for his notes and for providing the electrodes that I used in the project, as well as Dr. Weaver for writing the material that I used for the neuroscience section. I would also like to thank Brian Mhatre for continuing to let me use the Circuitlab account that I used to create the circuit diagrams. Thank you to all of the students at the Maker Faire who tested the game, even though it required me to put stick electrode gel on their heads. Lastly, I would like to thank Sonya Ledebeva for being the person who, when I was at a loss for what game could be controlled using only blinking, suggested Flappy Bird.

Bibliography

[1] https://www.instructables.com/id/DIY-EEG-and-ECG-Circuit/, Website accessed 2/10/19.
[2] William Lu, Evolution of Video Game Controllers (https://web.stanford.edu/group/htgg/sts145papers/wlu_2003_1.pdf)
[3] https://latimesblogs.latimes.com/technology/2009/06/microsofte3.html, Website accessed 2/10/19.
[4] http://www.asobostudio.com/games/fragments, Website accessed 2/10/19.
[5] https://www.wired.com/2015/01/intel-gave-stephen-hawking-voice/, Website accessed 2/10/19.
[6] https://www.britannica.com/biography/Luigi-Galvani#ref2564, Website accessed 2/10/19
[7] https://aim25.com/cats/9/10190.htm, Website accessed 2/10/19.
[8] Neurological Stamp, Journal of Neurology, Neurosurgery and Psychiatry (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1738204/pdf/v074p00009.pdf).
[9] https://www.mcgill.ca/about/history/penfield, Website accessed 2/10/19.
[10] S. Bozinovski et. al., Using EEG alpha rhythm to control a mobile robot, Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 4-7 Nov. 1988 (https://ieeexplore.ieee.org/document/95357)
[11] Linxing Jiang et. al., BrainNet: A Multi-Person Brain-to-Brain Interface for Direct Collaboration Between Brains, arXiv, 23 Sep. 2018 (https://arxiv.org/pdf/1809.08632.pdf).
[12] Dr. Dann's notes.
[13] http://bio1152.nicerweb.com/Locked/media/ch48/axon.html, Website accessed 3/21/19.
[14] http://learn.neurotechedu.com/oscillations/, Website accessed 3/21/19.
[15] https://www.ncbi.nlm.nih.gov/books/NBK390343/, Website accessed 3/21/19.
[16] https://www.spinalcord.com/occipital-lobe, Website accessed 3/21/19.
[17] https://www.psych.westminster.edu/psybio/BN/Labs/Brainwaves.htm, Website accessed 2/10/19.
[18] https://www.electronics-tutorials.ws/opamp/opamp_1.html, Website accessed 3/21/19.
[19] https://www.electronics-tutorials.ws/opamp/opamp_2.html, Website accessed 3/21/19.
[20] https://www.electronics-tutorials.ws/opamp/opamp_5.html, Website accessed 3/21/19.

[21] https://www.electronicdesign.com/power/what-s-difference-between-operational-amplifiers-and-instrumentation-amplifiers, Website accessed 2/10/19.
[22] https://www.edn.com/electronics-blogs/bakers-best/4312741/Understanding-CMR-and-instrUmentation-amplifiers, Website accessed 2/10/19.
[23] https://www.analog.com/media/en/technical-documentation/data-sheets/AD620.pdf
[24] Paul Scherz, Practical Electronics for Inventors, McGraw-Hill Education TAB, 2013.
[25] http://www.opensourceinstruments.com/Electronics/A3013/HTML/Hum.html, Website accessed 2/10/19.
[26] https://github.com/sourabhv/FlapPyBird, Website accessed 5/15/19.

Appendix: Code Used In Project

```
import serial
import datetime
import subprocess
import pyautogui
import time

from pynput.keyboard import Listener

ser = serial.Serial('/dev/ttyACM0', 9600)

old_time = datetime.datetime.now()

# focus the flappy bird window
subprocess.Popen(["wmctrl", "-a", "Flappy Bird"])


restarting = False
calibrating = False
CUTOFF_STEPPING = 2  # the amount the cutoff is adjusted by each iteration during
calibration


def start_calibration(key):
    global calibrating
    listener.stop()
    calibrating = True


def end_calibration(key):
    global calibrating
    listener.stop()
    calibrating = False
```

```
def calibrate():
    global restarting
    global listener
    global calibrating

    restarting = False

    listener = Listener(on_press=start_calibration)


    # Listen for the user to press a key.
    listener.start()


    while not calibrating:
        # Wait. Nothing will happen until the user presses a key and begins calibration.
        time.sleep(0.1)

    print("calibration begun")

    # listen for the user to end calibration by pressing a key.
    listener = Listener(on_press=end_calibration)
    listener.start()

    data = []
    while calibrating: # continue reading as long as calibration is still continuing
        try:
            # read a value from the Arduino and append it to the data array.
            read = int(ser.readline().decode("utf-8").strip("\n").strip("\r"))
            data.append(read)
        except UnicodeDecodeError:  # happens very rarely when input is read in the middle of
being printed
            print("problem")
        except ValueError:  # shouldn't happen, but if it does it's probably fine to ignore.
            print("Value error")

    print("calibration ended")

    lowest_cutoff = None  # the lowest cutoff value that detects the correct number of blinks
    highest_cutoff = None  # the highest cutoff value that detects the correct number of blinks
    n_above_cutoff = 0  # the number of consecutive readings above the cutoff

    cutoff = 600

    # The following while loop helps to identify a cutoff value to be used for detecting a blink.
    # Given the number of times the user blinks during calibration, the loop finds the highest
    # and lowest cutoff value that successfully detects exactly that number of blinks.
    while True and cutoff < 1024:  # if cutoff gets to over 1024 something clearly failed.
        blinks = 0
        currently_in_blink = False  # whether we are currently in a blink
        for read in data:
            if read > cutoff and not currently_in_blink:  # a blink might have begun.
                n_above_cutoff += 1
            if n_above_cutoff > 8:  # if we have at least 9 consecutive datapoints above the cutoff, a
blink has begun.
                n_above_cutoff = 0
                currently_in_blink = True
            if currently_in_blink and read < cutoff:  # a blink is over when values drop below the
cutoff
                blinks += 1
                currently_in_blink = False
        if lowest_cutoff:  # we already have a lower bound cutoff
            if blinks != 4:  # if our cutoff is now too high to get 4 blinks, we are done. Use the last
cutoff as the
                # upper bound.
                highest_cutoff = cutoff - CUTOFF_STEPPING
                break
        else:
            if blinks == 4:
                lowest_cutoff = cutoff
        cutoff += CUTOFF_STEPPING

    if cutoff >= 1024:
        print("error")

    # The lower end of the cutoff will likely end up detecting a blink that isn't there. The upper
end probably won't
    # detect some blinks. So the middle of the two is used.
    cutoff = (highest_cutoff - lowest_cutoff)/2 + lowest_cutoff

    print("cutoff: " + str(cutoff))
    run_game(cutoff)


def restart_calibration(key):
    global restarting
    global listener
    try:
        if key.char == "c":  # this works to reset calibration when the c key is pressed
            print("got in")
            restarting = True
            listener.stop()
    except AttributeError:
        pass


def run_game(cutoff):
    # now that a cutoff has been identified, we use it to detect blinks
    global restarting
    global listener
    print("game going on")
    listener = Listener(on_press=restart_calibration)
    listener.start()
    blinks = 0
    n_above_cutoff = 0
    currently_in_blink = False

    while True:
        if restarting:
            calibrate()
        try:
            read = int(ser.readline().decode("utf-8").strip("\n").strip("\r"))
            if read > cutoff and not currently_in_blink:
                n_above_cutoff += 1
            if n_above_cutoff > 8:
                n_above_cutoff = 0
                currently_in_blink = True
            if currently_in_blink and read < cutoff: # blink is over
                pyautogui.keyDown("space")
                pyautogui.keyUp("space")
                blinks += 1
                currently_in_blink = False
        except UnicodeDecodeError:
            print("problem")
        except ValueError:
            print("Value error")


calibrate()
```