A Low-Cost Modular EEG System and Software for Educational and Prototyping Applications

Ryan Corp

October 10, 2025

Abstract

Brain-computer interfaces (BCIs) offer a wealth of educational value and research potential; however, the high cost of EEG hardware limits accessibility to beginners, hobbyists, students, and independent researchers. This project presents a low-cost modular EEG data acquisition system designed for educational and prototyping purposes. The system is built around three-electrode modules featuring AD620 instrumentation amplifiers, ADS1115 analog-to-digital converters, and operates on a single Raspberry Pi Zero v1.3. Each module is capable of recording signals from either a single referential site or a pair of electrodes in a bipolar configuration. By utilizing four modules, a single Raspberry Pi can record up to five bipolar channels with low latency. Custom Python code performs bandpass filtering, artifact interpolation, USB communication, and real-time visualization. It also supports near-real-time Hjorth parameter visualization and power spectral analysis. Tests using a breadboard prototype indicate that the system can reliably process, capture, and visualize EEG signals, and offers an accessible and engaging introduction to neuroengineering and brain–computer interfaces.

1. Background

Electroencephalography (EEG) datasets are widely available online for use by hobbyists and independent researchers; however, opportunities for direct, hands-on data collection are limited to research centers and academic institutions. The prohibitive cost of entry-level EEG hardware and the lack of well-documented low-cost alternatives are significant obstacles to both learning and experimentation. This project addresses that gap, providing students, hobbyists, and independent researchers with an accessible introduction to BCI hardware design, signal acquisition, and digital preprocessing.

Commercial platforms, such as OpenBCI, have made biosensing technologies more accessible; however, even their most basic kits remain prohibitively expensive. For example, the OpenBCI four-lead Biosensing Starter Bundle costs over one thousand dollars [1]. In contrast, the circuit described in this paper can be assembled to utilize the same number of channels, either in bipolar or referential montage, for under two hundred dollars, and with further development, could be constructed for even less.

This system draws inspiration from prior open-source EEG amplifier designs, most notably those by Chip Epstein [2] and Cah6 [3]. Epstein's cost-effective, Arduino-based circuit has seen widespread use in several well-documented entry-level projects [4], [5], [6]. While his system minimizes cost and complexity, it struggles with limited sample resolution, lacks line-frequency filtering, and only implements basic waveform visualization. Alternatively, Cah6's EEG, while providing ample software support and an in-depth tutorial, uses multiple layers of analog filtering, which present sourcing, complexity, and cost issues [3], [4]. Related efforts, such as those in [5] and [6], provide either detailed circuit documentation or extensive software pipelines, though none fully integrate both aspects.

Building on the strengths of prior efforts, the design presented in this paper emphasizes cost-effectiveness, modularity, sampling resolution, and an educational workflow that connects circuit-level signal acquisition with digital preprocessing and feature extraction. As such, it serves as an introduction for those seeking to understand the hardware and analytical foundations of EEG signal processing.

2. System Overview

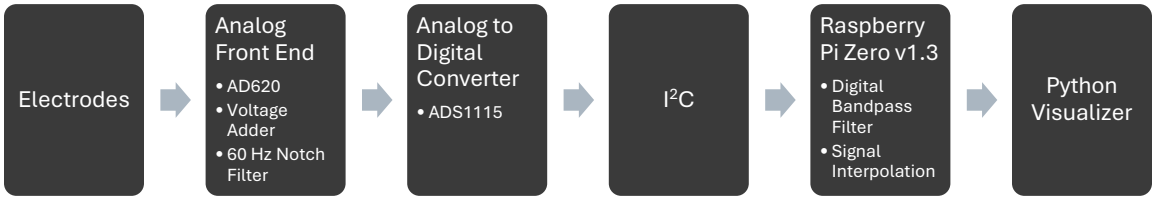


Figure 1: Diagram of the complete signal chain.

2.1 Signal Acquisition

Each module in this low-cost EEG system can operate in referential or bipolar recording modes. A reference electrode is connected to a common ground site, typically located at A1 or A2 in the international 10–20 placement system [7]. This same reference electrode can be used across multiple modules in a multi-module configuration.

2.2 Analog Front End

The analog circuit can generate either a referential or a bipolar signal depending on its configuration [7], [8, pp. 42-45]. Each module uses an AD620 instrumentation amplifier [9] for low-noise differential amplification.

In a referential configuration, one input pin of the AD620 instrumentation amplifier [9] is connected to an active lead, while the other is linked to the reference electrode. Alternatively, when configured to take a bipolar signal, two active leads are directly connected to an AD620's input pins. Due to the sequential nature of this type of montage, when using multiple modules, each successive AD620 shares one lead with the preceding module in the series.

After amplification, the signal is conditioned through a voltage adder circuit to center the waveform within the ADC's input range and provide additional gain. A 60 Hz notch filter is subsequently applied to mitigate line-frequency interference.

2.3 Analog-to-Digital Conversion

To digitize the conditioned analog signal, it is passed to an ADS1115 [10], a 16-bit analog-to-digital converter (ADC) breakout board from Adafruit [11]. The ADS1115 integrates a programmable gain amplifier and supports up to four channels on a single I²C bus via selectable address pins [12]. With a maximum sample rate of 860 samples per second (SPS), the ADC comfortably exceeds the Nyquist frequency of standard EEG signals [8, p. 24], [13].

2.4 Data Acquisition and Processing

The ADC output is transmitted via I^2C to a Raspberry Pi Zero v1.3 [14], which performs data acquisition and preprocessing using Python. The incoming data is segmented into fixed-length chunks and filtered using a fourth-order Butterworth bandpass filter between 1 and 50 Hz. This is followed by mild artifact correction via interpolation of large voltage spikes nearing the ADC's readable range (excerpts shown in Appendix D). The preprocessed signal is then transmitted via USB to a host computer for storage, plotting, and feature extraction in Python.

3. Hardware Design

This low-cost EEG acquisition system is a modular unit with a single reference electrode common to all modules. Each unit interfaces with a single instrumentation amplifier and a 16-bit analog-to-digital converter. Using the analog-to-digital converter's variable addresses, a single Raspberry Pi Zero v1.3 can aggregate the signals from all modules for preprocessing. Figure 1 illustrates the signal chain for a single module, showing the path from the electrode input to the preprocessed digital output. For replication purposes, the complete circuit schematic for a single module and a constructed breadboard example are provided in Appendices B and C, respectively.

3.1 Design Considerations

The system draws heavily from open-source EEG amplifier designs by C. Epstein and cah6. While Epstein's Arduino-based design offers a beginner-friendly and affordable solution, it provides only 10-bit resolution and lacks digital preprocessing. This design replaces the Arduino with a Raspberry Pi Zero v1.3 and an ADS1115 ADC, improving signal resolution to 16 bits. It also adds a 60 Hz filter, as seen in cah6's design, to reduce line-frequency interference. The design utilizes the low-cost LM358 dual operational amplifiers (op-amps). Although CA3130 operational amplifiers [15] were initially considered for their lower noise, they were excluded due to their higher cost, twelve times that of the chosen op-amp [16], [17], [18].

3.2 Circuitry

Electrode signals are first routed through an AD620 instrumentation amplifier with a gain of 17.47x, providing low-noise differential amplification. A voltage adder then applies an additional amplification of 66.67 times and centers the signal within the ADC's input range, yielding a total circuit gain of approximately 1164 times. This gain amplifies typical EEG signals from the ±10–100 µV range to within ±0.116 V. A 10 µF capacitor placed before the voltage adder creates an approximately 0.016 Hz high-pass filter, which removes DC bias that could disrupt the adder's signal centering while preserving EEG frequencies.

The circuit's total gain is determined by the AD620 and the voltage adder. The AD620 has a gain defined by:

$$G_{AD620} = 1 + \frac{49.4\ k\Omega}{R_6}$$

The gain from the voltage adder is defined as:

$$G_{adder} = \frac{R_{10}}{R_5}$$

Values and the locations of R_{10}, R_6 and R_5 can be found in the circuit diagram in Appendix B. To maintain the existing high-pass filter while adjusting gain, it is recommended to change R_6 rather than R_5.

To suppress line-frequency interference, a 60 Hz Sallen-Key notch filter is incorporated at the voltage adder output. The filtered signal then passes through a small voltage protection circuit before reaching the ADC. This protection is necessary as the analog front end can produce voltages up to 5 V, while the ADS1115 ADC must operate within 3.3 V rails to match the Raspberry Pi Zero's logic levels. According to the ADS1115 datasheet, its maximum input voltage is defined as:

$$V_{max} = VDD + 0.3$$

In this configuration, the maximum input voltage is 3.6 V, which is below the analog front end's potential 5 V output. Therefore, the protection circuit uses a Schottky diode to shunt voltage spikes exceeding 3.3 V to the microcontroller.

The ADC's internal programmable gain amplifier (PGA) is set to 16, which limits the input range to 0-0.256 V in single-ended mode. The voltage adder circuit centers the signal at 0.128 V, coupling with the expected ±0.116 V amplified EEG signals to maximize the ADC's dynamic range.

When adjusting the voltage adder bias, the 60 Hz notch filter should be temporarily disconnected. Leaving it connected distorts the offset voltage, making accurate adjustment impossible.

3.3 Prototyping Limitations

The breadboard-based prototype used for this proof of concept demonstrated an elevated noise floor and significant artifacts due to unsoldered EEG leads losing connection under tension. A soldered or PCB implementation could substantially reduce coupling noise, loose connections, and provide more stable grounding.

3.4 Operation Notes:

- Avoid operating the system while a connected laptop is charging. In many USB power supplies, improper isolation introduces interference, leading to severe waveform instability. The author of this paper was unable to operate the device using a MacBook Air and other laptops due to this issue.
- The circuit should be used only with a desktop computer (which has grounded USB ports) or with a laptop running on battery power.

4. Data Acquisition and Preprocessing

In each module, the digitization and transmission of EEG signals are performed using an ADS1115 16-bit ADC, which communicates with a Raspberry Pi Zero v1.3 via the I^2C interface. Each recording module connects a single amplified signal to the ADS1115's A0 input pin, operating in single-ended, continuous conversion mode [11, pp. 10, 14]. For the single-module system built in this paper, the default I^2C address (0x48) was used. The SDA and SCL lines were connected directly to the corresponding Raspberry Pi pins, and communication was established using the Adafruit ADS1X15 Python library [19].

Running the ADS1115 in continuous conversion mode reduces latency by allowing the Raspberry Pi to immediately read the most recent sample processed by the ADS1115, simplifying programming. Although given the ADC sample rate of 860 SPS, well above the Nyquist frequency of EEG signals, it may be possible to use a single ADS1115 in single-shot conversion mode to operate up to four EEG modules with a usable sample rate of approximately 190-215 SPS.

The acquired data are stored in a rolling five-second buffer on the Raspberry Pi. From this buffer, a 150 ms segment is selected for preprocessing, with an additional 50 ms of padding added to both ends. This padding mitigates edge artifacts during the subsequent digital filtering step. These buffer management and timing parameters are implemented in i2c_read_loop_and_preprocessing.py (excerpts shown in Appendix D).

Digital filtering is performed by Python's scipy.signal.butter, implementing a fourth-order Butterworth bandpass filter with a bandwidth of 1 Hz to 50 Hz. Python's scipy.signal.filtfilt then runs this filter in both the forward and reverse directions to minimize phase shifts. The chosen 1 to 50 Hz range isolates the primary EEG frequency bands and rejects high-frequency noise. Post-filtering, data points exceeding ±0.126 V, near the limits of the ADC input range, are treated as artifacts and replaced with interpolated values.

After preprocessing, the padded regions are removed, and the cleaned 150 ms data segment is transmitted to a host computer over a USB serial connection. This connection provides input to two real-time feature extraction and visualization scripts implemented in Python.

5. Feature Extraction and Visualization

Once the preprocessed EEG data reaches the host computer, it is processed, visualized, and stored using the custom Python program eeg_reader.py (excerpts shown in Appendix D). On launch, the program creates a new CSV file in the working directory and scans for a serial connection to the Raspberry Pi. Each incoming voltage sample is adjusted by the theoretical amplification factor of 1164.44x to recover the original microvolt-scale signal. It is then recorded alongside its timestamp relative to the first sample received since the program was started.

Data visualization is achieved in real time using Matplotlib's plt.ion() interface, which displays three concurrent plots (Appendix E):

1. A cumulative plot of signal amplitude versus time for all recorded data.
2. A three-second window displaying the most recent data.
3. A detailed view of the most recently received chunk.

By storing the data in a CSV file, it can be accessed by hjorth_params_and_fft_viewer.py (excerpts shown in Appendix D), which performs real-time feature extraction and frequency analysis. The program uses a fixed-length buffer implemented with Python's collections.deque, preventing plotting latency from increasing as the CSV file grows.

The feature extraction process operates on three-second data segments, updating every 0.2 seconds for near real-time analysis. Data is analyzed in 64-sample chunks, which are converted from volts to microvolts and passed to a function that computes the Hjorth parameters, Activity, Mobility, and Complexity [20]:

- Activity represents the variance and overall power of the EEG signal.
- Mobility represents the mean frequency by evaluating the standard deviation of the first derivative of activity relative to the original signal.
- Complexity describes the deviation of the signal from a pure sine wave. A value of 1 indicates a sinusoidal signal, while higher values correspond to more irregular signals.

As stated above, to calculate Hjorth parameters, it is necessary to take the derivative of the data. Leveraging the uniform sample spacing, derivatives are determined using NumPy's diff() function. hjorth_params_and_fft_viewer.py then plots these values continuously over the last ten seconds.

Additionally, hjorth_params_and_fft_viewer.py calculates the power spectral density (PSD) of each chunk using the Fast Fourier Transform (FFT). The FFT is calculated via scipy.fft.rfft, and the corresponding frequencies with scipy.fft.rfftfreq. The resulting power distribution is displayed as a bar chart representing bandpower per frequency.

Combining these visualizations enables users to observe EEG signal features in both the time and frequency domains, allowing for real-time analysis and interpretation. Screenshots of the frequency-domain and Hjorth parameter visualizations are provided in Appendix E.

6. Discussion

6.1 Performance Notes

Proof-of-concept tests using the module described in this paper demonstrate that the device and preprocessing can isolate neural activity and suppress baseline drift (Appendix E). Recognizable signals were acquired from a participant in a referential configuration. As stated in the prototyping limitations section, although data were acquired from a breadboard-based implementation, the lack of soldered connections and the tension in the EEG leads caused frequent disconnections, signal loss, and significant artifacts, which resulted in voltage spikes and clipping in the visualized signal.

6.2 Future Work

Multiple opportunities exist to continue developing this prototype and accompanying software. Immediate improvements include using a soldered PCB to stabilize connections and substituting the LM358 amplifiers with a lower-noise alternative, such as the CA3130 or OPA333. Other potential improvements include replacing the voltage splitter and electrode-based ground with a proper rail splitter, such as the TLE2426; furthermore, one could replace the AD620 with the lower-priced, and potentially more precise, INA333. As a priority, the software package should be amended to support the proposed multi-module montages, and a fully realized four-module prototype device should be constructed, both using a single ADS1115 and the four ADS1115 designs to compare efficacy. Finally, to improve ease of use and plug-and-play functionality, the current Python scripts could be consolidated into a single graphical interface that supports configurable filtering parameters, adjustable plots, and enhanced data logging.

7. Conclusion

This project offers a low-cost, beginner-friendly EEG hardware and processing pipeline for students, hobbyists, and independent researchers. It provides an open-source framework for signal acquisition, filtering, and feature extraction. Beyond a simple prototype, this system offers low-cost, well-documented hardware and software to lower barriers to entry and increase

public participation in BCI research. This work, with its numerous opportunities for refinement and expansion, provides a foundation for future contributors to advance both the accessibility and development of neuroengineering technologies without requiring extensive technical knowledge.

Appendices

Appendix A

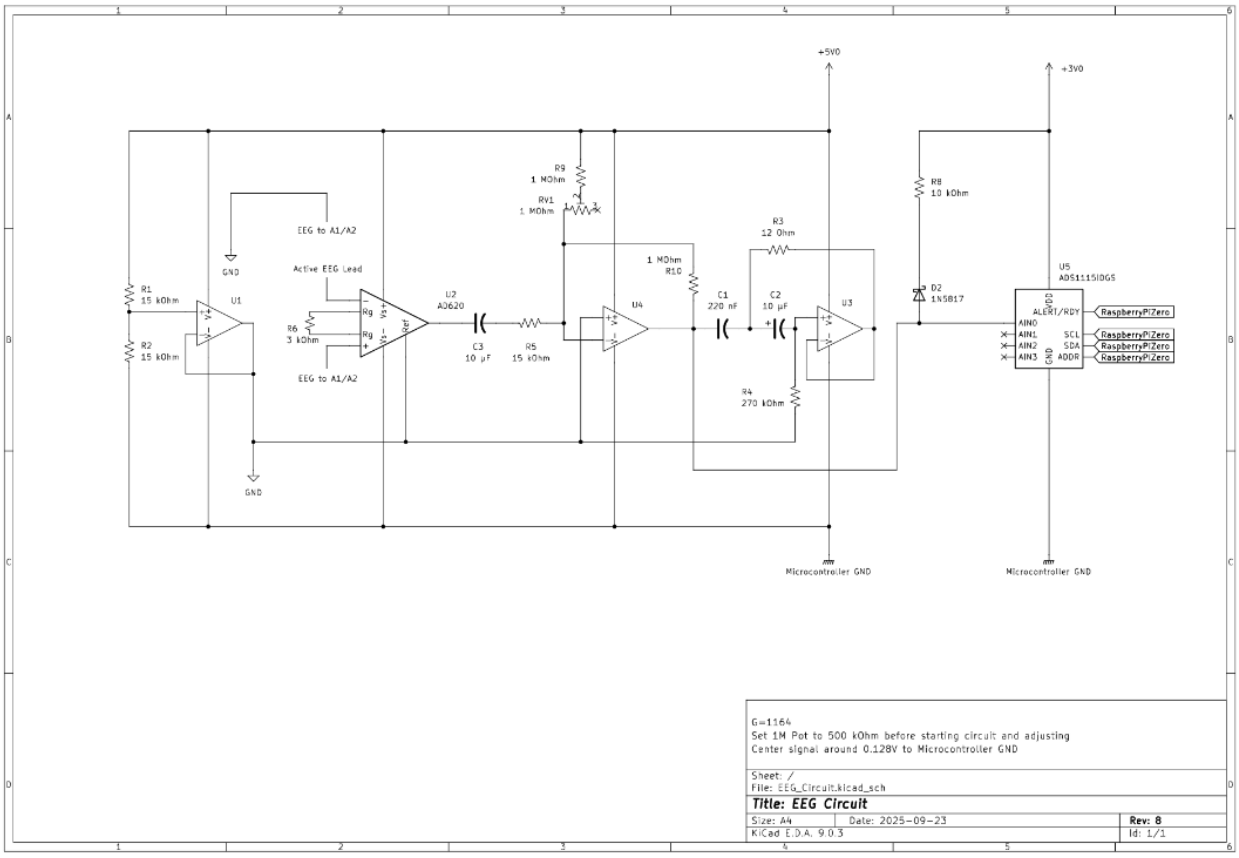
Bill of Materials

Material	1-2 Lead System	4 Lead bipolar system
Microcontroller – Raspberry Pi Zero v1.3	1	1
Analog-to-Digital Converter – ADS1115 (STEMMA QT/Qwiic)	1	1-4
Instrumentation Amplifier – AD620	1	4
Ceramic Capacitor – 10 µF	1	4
Aluminum Electrolytic Capacitor – 10 µF	1	4
Tantalum Capacitor – 0.22 µF	1	4
Schottky Diode – 1N5817	1	4
Operational Amplifier – LM358P	3	12
Resistor – 1 MΩ ¼W ±1%	2	8
Resistor – 270 kΩ ¼W ±1%	1	4
Resistor – 15 kΩ ¼W ±1%	3	6
Resistor – 10 kΩ ¼W ±1%	1	4
Resistor – 3 kΩ ¼W ±1%	1	4
Resistor – 12 Ω ¼W ±1%	1	4
Potentiometer – 1 MΩ	1	4
Electrode Lead Wire	2-3	6

Appendix B

Circuit Schematic

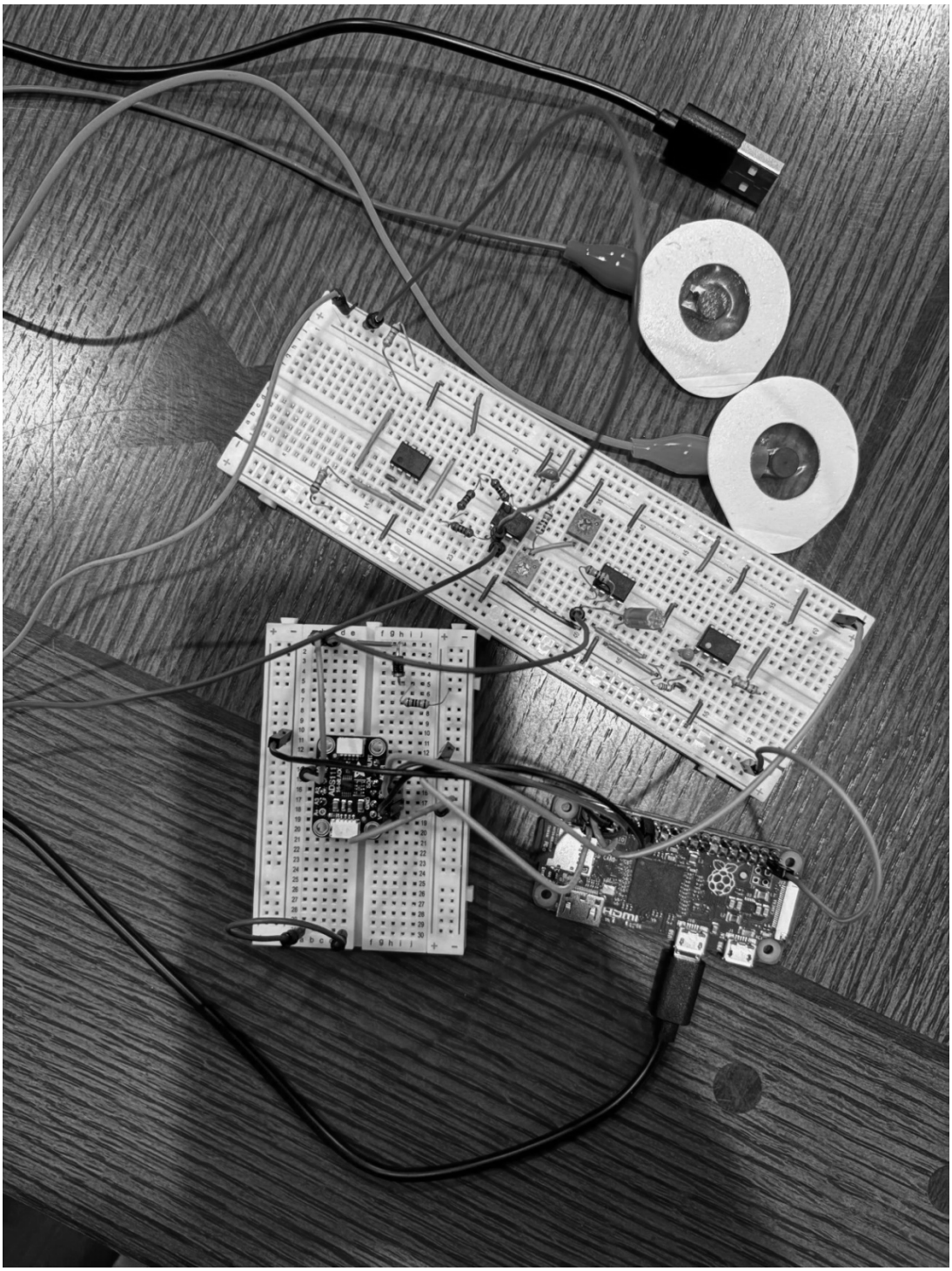
A full-sized PDF of the circuit diagram is available at https://github.com/ryancorp/EEG-Portfolio/.



Appendix C

Breadboard Image

A full-color image of the circuit schematic is available at https://github.com/ryancorp/EEG-Portfolio/.



Appendix D

The Python scripts used in this project are available at https://github.com/ryancorp/EEG-Portfolio/.

Data storage and visualization logic (Excerpt from: eeg_reader.py)

```
SAMPLE_RATE_HZ = 860 #SPS
SAMPLE_INTERVAL = 1.0 / SAMPLE_RATE_HZ #Seconds per sample
SAMPLE_RANGE = 0.256 #Max voltage range for ADC PGA

# Number of samples per chunk (must match Raspberry Pi CHUNK_SIZE)
CHUNK_DURATION_SECONDS = 0.15
CHUNK_SIZE = int(SAMPLE_RATE_HZ * CHUNK_DURATION_SECONDS)

CIRCUIT_GAIN = 1164.44 #Circuit gain

CSV_FILE = "eeg_data.csv"

GRAPH_BOUNDS = SAMPLE_RANGE/(2*CIRCUIT_GAIN) # Expected voltage range

# Check if the file exists, then delete it
if os.path.exists(CSV_FILE):
    os.remove(CSV_FILE)
    print(f"{CSV_FILE} has been deleted.")
else:
    print(f"{CSV_FILE} does not exist.")

# Setup real-time plot
plt.ion()
fig, ax = plt.subplots(3, 1, figsize=(10, 8), sharex=False)

#Full history plot
line, = ax[0].plot([], [])
ax[0].set_ylim(-GRAPH_BOUNDS, GRAPH_BOUNDS)
ax[0].set_xlim(0, CHUNK_DURATION_SECONDS)
ax[0].set_xlabel('Time (s)')
ax[0].set_ylabel('Amplitude (V)')
ax[0].grid(True)
```

```
# Last 3 seconds plot
line_last, = ax[1].plot([], [])
ax[1].set_ylim(-GRAPH_BOUNDS, GRAPH_BOUNDS)
ax[1].set_xlim(0, 3)
ax[1].set_title("Last 3 Seconds")
ax[1].set_ylabel("Amplitude (V)")
ax[1].grid(True)

# Current chunk plot
line_chunk, = ax[2].plot([], [])
ax[2].set_ylim(-GRAPH_BOUNDS, GRAPH_BOUNDS)
ax[2].set_xlim(0, CHUNK_DURATION_SECONDS)
ax[2].set_title("Current Chunk")
ax[2].set_xlabel("Time (s)")
ax[2].set_ylabel("Amplitude (V)")
ax[2].grid(True)

fig.tight_layout()

# Store signal data
time_data = []
signal_data = []

t_chunk = np.linspace(0, CHUNK_DURATION_SECONDS, CHUNK_SIZE, endpoint=False)

# Finds serial ports to improve device compatability
def find_serial_port(retries=5, delay=2):
    keywords = ['raspberry', 'usb serial', 'usbmodem', 'ttyusb', 'ch340', 'cp210', 'ftdi']

    for attempt in range(retries):
        ports = list(serial.tools.list_ports.comports())
        if not ports:
            print(f"No serial ports found (attempt {attempt+1}/{retries})...")
            time.sleep(delay)
            continue

   for port in ports:
            desc = port.description.lower()
```

```
            device = port.device.lower()
            hwid = port.hwid.lower()

            # Match on device description or hardware ID
            if any(k in desc or k in device or k in hwid for k in keywords):
                print(f"Found Raspberry Pi (or similar device): {port.device} — {port.description}")
                return port.device

        print(f"No matching device found (attempt {attempt+1}/{retries})...")
        time.sleep(delay)

    raise RuntimeError("Could not detect Raspberry Pi serial connection.")

# Read chunk sent over serial
def read_chunk(ser, chunk_size):
    num_bytes = chunk_size * 4  # 4 bytes per float
    data = b''
    while len(data) < num_bytes:
        data += ser.read(num_bytes - len(data))

    # Convert binary data to floats
    chunk = struct.unpack(f'{chunk_size}f', data)
    #print(f"Chunk {chunk_index} received: {chunk}")
    return np.array(chunk)/CIRCUIT_GAIN

# Appends chunk to created CSV. This CSV will be overwritten on startup so move somewhere
else if you would like to save it
def append_chunk_to_csv(time_array, signal_array, filename=CSV_FILE):
    file_exists = os.path.exists(filename)

    with open(filename, mode='a', newline='') as f:
        writer = csv.writer(f)
        if not file_exists:
            writer.writerow(["time", "signal"])  # write header only if file is new
        for t, s in zip(time_array, signal_array):
            writer.writerow([t, s])
```

Serial communication and buffering loop
(Excerpt from: i2c_read_loop_and_preprocessing.py)

```
SAMPLE_RATE_HZ = 860 #SPS
SAMPLE_INTERVAL = 1.0 / SAMPLE_RATE_HZ #Seconds per sample

SLIDING_BUFFER_SECONDS = 5
SLIDING_BUFFER_SIZE = SLIDING_BUFFER_SECONDS * SAMPLE_RATE_HZ

CHUNK_DURATION_SECONDS = 0.15 #Amount of data sent per step size
CHUNK_SIZE = int(SAMPLE_RATE_HZ * CHUNK_DURATION_SECONDS)

PAD_DURATION_SECONDS = 0.05 #Seconds to pad around data (adds a delay to data received)
PAD_SIZE = int(SAMPLE_RATE_HZ * PAD_DURATION_SECONDS)

i2c = busio.I2C(board.SCL, board.SDA)

ser = serial.Serial('/dev/serial0', 115200)

lock = Lock()

# Create the ADS object and specify the gain
ads = ADS.ADS1115(i2c)
ads.gain = 16
ads.data_rate = 860
chan = AnalogIn(ads, ADS.P0)

# Serialize chunk as binary
def serialize_chunk_binary(chunk):
    return struct.pack(f'{len(chunk)}f', *chunk)

# Worker thread to filter and send chunks
def process_and_send():
    global active_buffer
    prev_pad = None


  while True:
        lock.acquire()
```

```
        buffer_to_process = active_buffer.copy()
        lock.release()

        #Chunk and padding logic to avoid bandpass edge artifacts
        try:
            raw_chunk = buffer_to_process[-(CHUNK_SIZE + PAD_SIZE):]
            if prev_pad is not None:
                chunk_with_pad = np.concatenate([prev_pad, raw_chunk])
            elif len(buffer_to_process) >= (CHUNK_SIZE + 2 * PAD_SIZE):
                chunk_with_pad = np.array(buffer_to_process)
            filtered = bandpass_filter(chunk_with_pad, 1, 50, SAMPLE_RATE_HZ)
            clean = interpolate_artifacts(filtered)
            latest_chunk = clean[PAD_SIZE:-PAD_SIZE]
            prev_pad = raw_chunk[-PAD_SIZE:]
            ser.write(serialize_chunk_binary(latest_chunk))
        finally:
            continue
        time.sleep(CHUNK_DURATION_SECONDS)  # throttle processing

# Start worker thread
active_buffer = deque(maxlen=SLIDING_BUFFER_SIZE)
thread = Thread(target=process_and_send, daemon=True)
thread.start()

# Main loop: continuously read ADC into the active buffer
sample_counter = 0
start_time = time.perf_counter()
while True:
    current_time = time.perf_counter()
    elapsed = current_time - start_time
    target_time = sample_counter * SAMPLE_INTERVAL
    sleep_time = target_time - elapsed
    if sleep_time > 0:
        time.sleep(sleep_time)


  # Read ADC sample
    sample = chan.voltage
```

```
lock.acquire()
active_buffer.append(sample)
lock.release()

sample_counter += 1
```

Bandpass filter and interpolation
(Excerpt from: i2c_read_loop_and_preprocessing.py)

```
def bandpass_filter(signal, low, high, fs, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [low / nyq, high / nyq], btype='band')
    return filtfilt(b, a, signal)

#Artifact rejection
def interpolate_artifacts(signal, threshold=0.252/2): #ADC clips at 0.253 V and 0 V
    signal = np.copy(signal)
    artifact_mask = np.abs(signal) > threshold

 if np.all(artifact_mask):
            return np.zeros_like(signal)

    good_idx = np.where(~artifact_mask)[0]
    bad_idx = np.where(artifact_mask)[0]

    #Linear interpolation
    interpolated = np.interp(bad_idx, good_idx, signal[good_idx])
    signal[10ad_idx] = interpolated

    return signal
```

CSV reading routine
(Excerpt from: hjorth_params_and_fft_viewer.py)

```
SAMPLE_RATE = 860
CSV_FILE = "../data_acquisition/eeg_data.csv"
NO_DATA_TIMEOUT = 5 #Seconds of no-data to turn off interactive plot
MAX_TIME_WINDOW = 10 #Seconds to display

WINDOW_SIZE = int(SAMPLE_RATE/2) # samples per window
STEP_SIZE = 64 # step size for moving window
POLL_INTERVAL = 0.2 # seconds between file checks
TAIL_SECONDS = 3 # read last N seconds of samples

def read_latest_samples_tail(filename, seconds=TAIL_SECONDS):
    if not os.path.exists(filename):
        return np.array([]), np.array([])
    n_rows = int(seconds * SAMPLE_RATE)
    # Use deque to keep last n_rows + header line
    try:
        with open(filename, 'r') as f:
            # keep header + last n_rows lines
            tail = deque(f, maxlen=n_rows + 1)
    except Exception:
        return np.array([]), np.array([])

    if not tail:
        return np.array([]), np.array([])

    # Ensure header is first line; if header not in deque, read header separately
    first_line = tail[0].strip()
    if not first_line.startswith("time"):
        # header likely not in tail -> read header from file start then combine last n rows
        with open(filename, 'r') as f:
            header = f.readline()
        text = header + "".join(list(tail))
    else:
        text = "".join(list(tail))

 try:
```

```
    df = pd.read_csv(io.StringIO(text))
    return df['time'].to_numpy(), df['signal'].to_numpy()
except Exception:
    return np.array([]), np.array([])
```

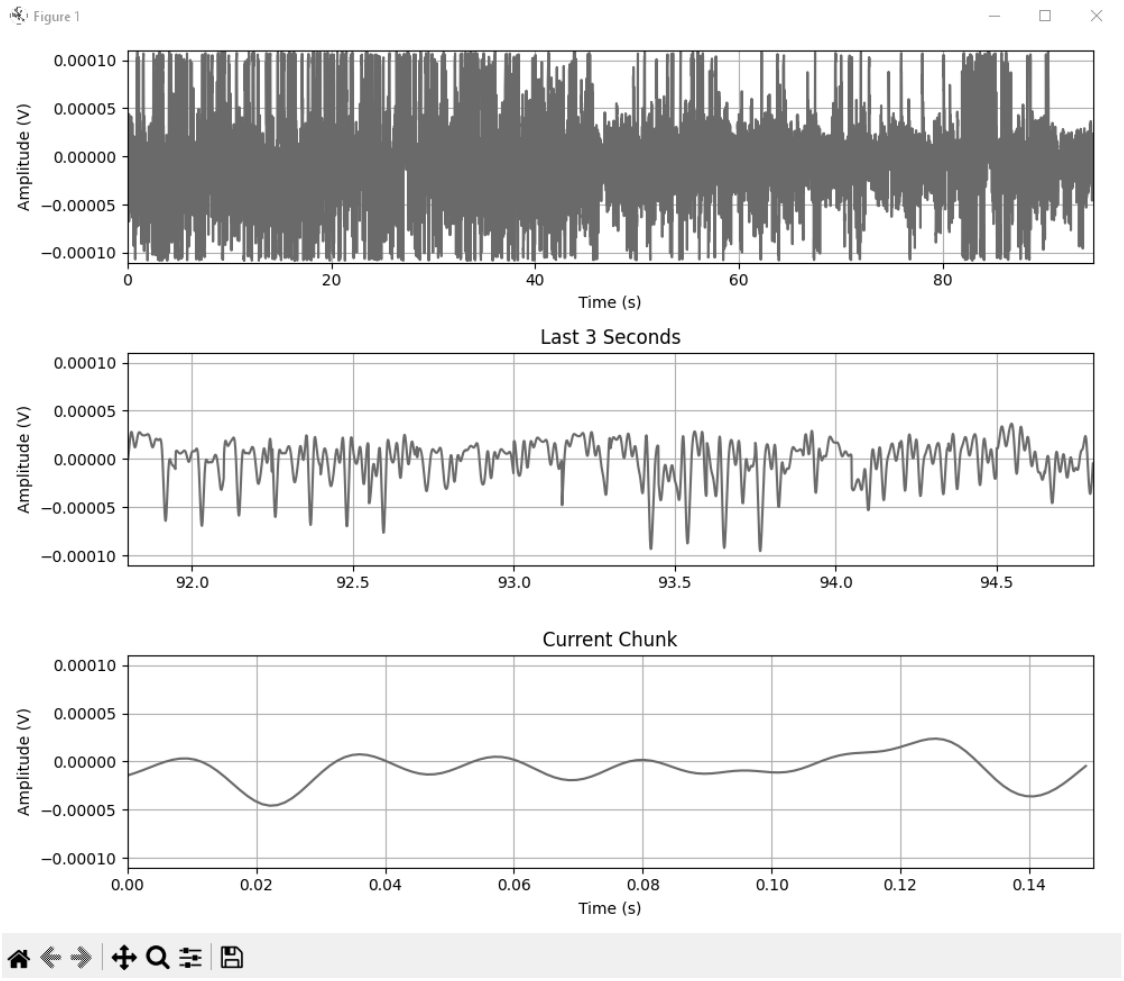
Hjorth parameter and FFT feature extraction routines
(Excerpt from: hjorth_params_and_fft_viewer.py)

```
# Computes Hjorth Params in microvolts^2 for Activity
def hjorth_parameters(signal):
    signal = signal * (10**6) #Convert uV signal to V

    if len(signal) < 2:
        return np.nan, np.nan, np.nan  # Not enough data

    # Activity
    activity = np.var(signal)

    # First derivative
    diff_signal = np.diff(signal)

    # Mobility
    mobility = np.sqrt(np.var(diff_signal) / activity)

    # Complexity
    diff2_signal = np.diff(diff_signal)
    mobility_diff = np.sqrt(np.var(diff2_signal) / np.var(diff_signal))
    complexity = mobility_diff / mobility

    return activity, mobility, complexity

# Computes FFT Power
def compute_fft(signal, fs=SAMPLE_RATE):
    signal = signal * (10**6) #Convert uV signal to V
    N = len(signal)
    yf = rfft(signal)
    xf = rfftfreq(N, 1/fs)
    psd = np.abs(yf)**2 / N  # Power spectral density
    return xf, psd
```

Appendix E

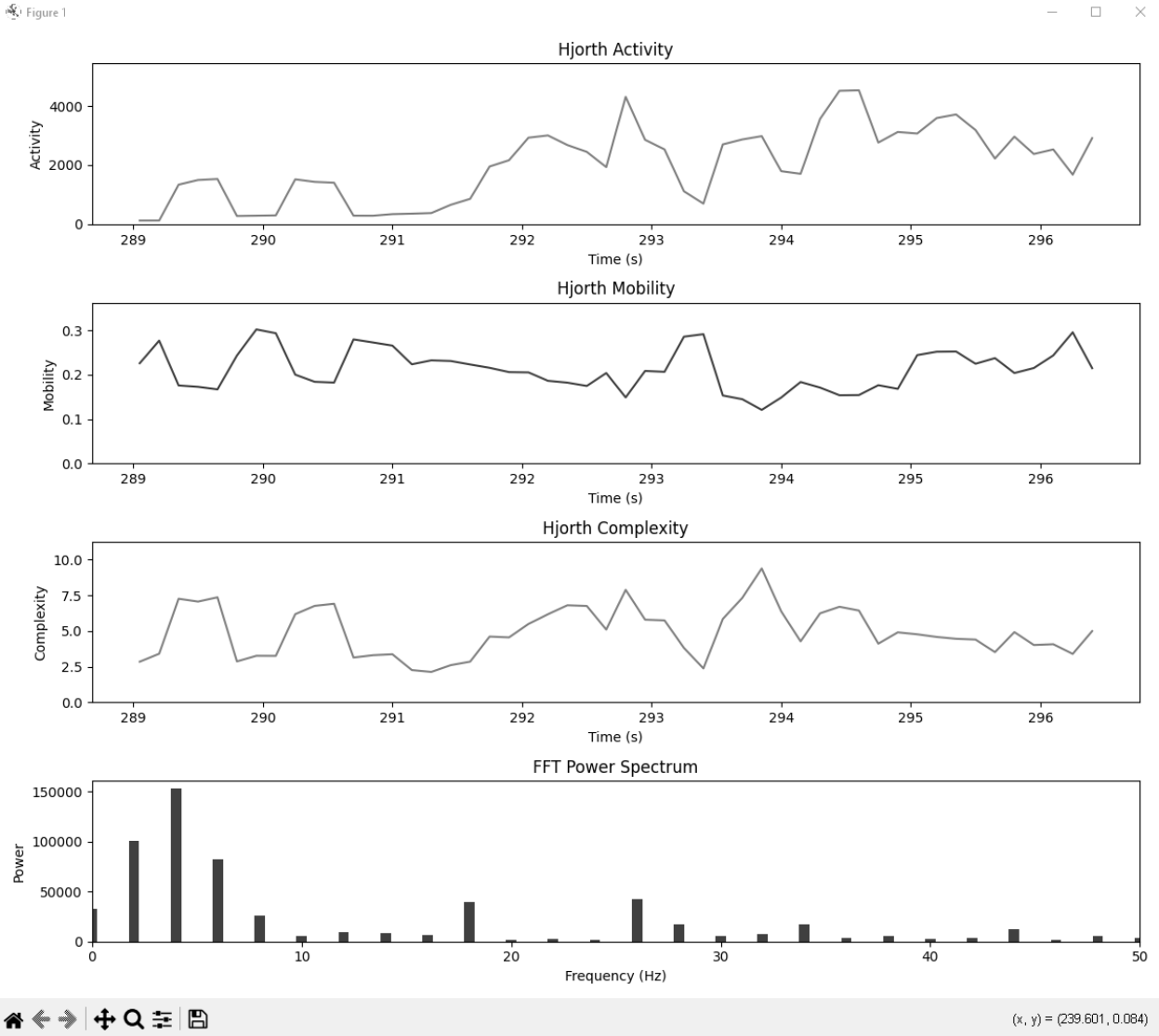
Signal Plot

A full-color image of the pictured waveform is available at https://github.com/ryancorp/EEG-Portfolio/.



Feature Plot

A full-color image of the feature plots below is available at https://github.com/ryancorp/EEG-Portfolio/.



References

[1] “Biosensing Starter Bundle,” *OpenBCI*, 2015. https://shop.openbci.com/products/biosensing-starter-bundle?srsltid=AfmBOooZW34DXjRMADrFQMKMT046x-OEp9Lo4T9CD59FJCSn6VKfGpQm (accessed Oct. 07, 2025).

[2] C. Epstein, “chipstein - Homebrew Do-it-yourself EEG, EKG, and EMG,” Mar. 28, 2022. https://sites.google.com/site/chipstein/homebrew-do-it-yourself-eeg-ekg-and-emg/ (accessed Jul. 30, 2025).

[3] cah6, “DIY EEG (and ECG) Circuit,” *Instructables*. https://www.instructables.com/DIY-EEG-and-ECG-Circuit/ (accessed Jul. 28, 2025).

[4] E. Ricker, “Final Project: DIY EEG, EEG: Cheap and Small,” *fab.cba.mit.edu*, Dec. 2015. https://fab.cba.mit.edu/classes/863.15/section.Harvard/people/Ricker/htm/Final_Project.html (accessed Jul. 29, 2025).

[5] Marblestone and Fracchia, “FabECG: a simple electrocardiogram board,” *fab.cba.mit.edu*. https://fab.cba.mit.edu/classes/863.12/people/Adam.Marblestone/AHM_week05.html (accessed 7AD).

[6] C. Moyes and M. Jiang, “Brain-Computer Interface Using Single-Channel Electroencephalography,” *people.ece.cornell.edu*, 2012. https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/cwm55/cwm55_mj294/index.html (accessed Jul. 30, 2025).

[7] L. Hirsch and R. Brenner, *Atlas of EEG in Critical Care*. John Wiley & Sons. Accessed: Oct. 10, 2025. [Online]. Available: https://www.ilae.org/files/ilaeBook/samplePages/Atlas%20of%20EEG%20Sample%20Pages%2012%20pg.pdf

[8] P. L. Nunez and Ramesh Srinivasan, *Electric fields of the brain: the neurophysics of EEG*. Oxford: Oxford University Press, 2006, pp. 24, 42–45.

[9] Analog Devices, “AD620 Low Cost Low Power Instrumentation Amplifier,” Rev. H, 2009. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD620.pdf

[10] Texas Instruments, “ADS1113, ADS1114, and ADS1115 Ultra-Small, Low-Power, 16-Bit ADCs With Internal Reference,” SBAS444B, May 2009, revised October 2009. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

[11] Adafruit Industries, ADS1115 16-Bit ADC - 4 Channel with Programmable Gain Amplifier - STEMMA QT / Qwiic, Product ID 1085. [Online].
Available: https://www.adafruit.com/product/1085

[12] L. Clark, “Adafruit 4-Channel ADC Breakouts,” *Adafruit Learning System*, Nov. 29, 2012. https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/assembly-and-wiring (accessed Jul. 30, 2025).

[13] B. Olshausen, “Aliasing, PSC 129 - Sensory Processes,” Redwood Center for Theoretical Neuroscience, Oct. 2000. Accessed: Oct. 10, 2025. [Online]. Available: https://www.rctn.org/bruno/npb261/aliasing.pdf

[14] Raspberry Pi Foundation, "Raspberry Pi Zero — Reduced Schematics," Nov. 16, 2016. [Online]. Available: https://datasheets.raspberrypi.com/rpizero/raspberry-pi-zero-reduced-schematics.pdf

[15] Renesas Electronics Corporation, “CA3130, CA3130A 15MHz, BiMOS Operational Amplifier with MOSFET Input/CMOS Output,” FN817, Rev.6.00 Aug 1, 2005. [Online].
Available: https://www.renesas.com/en/document/dst/ca3130-ca3130a-datasheet

[16] Texas Instruments, “Industry-Standard Dual Operational Amplifiers,” SLOS068AB, June 1979, revised October 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm358.pdf

[17] “CA3130EZ,” *DigiKey*. https://www.digikey.com/en/products/detail/renesas-electronics-corporation/CA3130EZ/1060748 (accessed Oct. 10, 2025).

[18] “LM358P,” *DigiKey*. https://www.digikey.com/en/products/detail/texas-instruments/LM358P/277042 (accessed Oct. 10, 2025).

[19] L. Clark, “Adafruit 4-Channel ADC Breakouts,” *Adafruit Learning System*, Jun. 29, 2025. https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/python-circuitpython (accessed Oct. 10, 2025).

[20] S.-H. Oh, Y.-R. Lee, and H.-N. Kim, “A Novel EEG Feature Extraction Method Using Hjorth Parameter,” *International Journal of Electronics and Electrical Engineering*, vol. 2, no. 2, pp. 106–110, Jun. 2014, doi: https://doi.org/10.12720/ijeee.2.2.106-110.

[21] A. Ortiz and J. Minguz, “Main features of the EEG amplifier explained,” *Bitbrain*, Apr. 03, 2020. https://www.bitbrain.com/blog/eeg-amplifier (accessed Jul. 29, 2025).

[22] “Summing Amplifier is an Op-amp Voltage Adder,” *Basic Electronics Tutorials*, Feb. 2019. https://www.electronics-tutorials.ws/opamp/opamp_4.html (accessed Jul. 29, 2025).

[23] R. Rager, “Tools of the Trade,” Jan. 2022. [Online]. Available: https://aset.org/wp-content/uploads/2022/01/10-20_System_Demonstration.pdf (accessed Jul. 29, 2025).

[24] “Connect to a Raspberry Pi Zero W via USB - No Mini HDMI Cable Needed,” *YouTube*, https://www.youtube.com/watch?v=xj3MPmJhAPU (accessed May 09, 2024).

[25] Mahmoodmustafashilleh, “How to Use ADS1115 With the Raspberry Pi (Part 1),” *Instructables*, https://www.instructables.com/How-to-Use-ADS1115-With-the-Raspberry-Pi-Part-1/ (accessed Jul. 29, 2025).