Researching on TGAM1 and how to analyze raw signal to Brainwave Band using SciPy library in Python

I. TGAM1: How does it work?

1. Hardware analysis.

TGAM is the product from NeuroSky that enables a device to interface with the wearer's brainwaves. It includes the sensor that touches the forehead, the contact and reference points on the ear pad, and the onboard chip that processes all the data and provides it to software and applications in digital form. Both raw brainwaves and the eSense Meters (Attention and Meditation) are calculated on the ThinkGear chip.

Figure 1: Top view picture of TGAM1

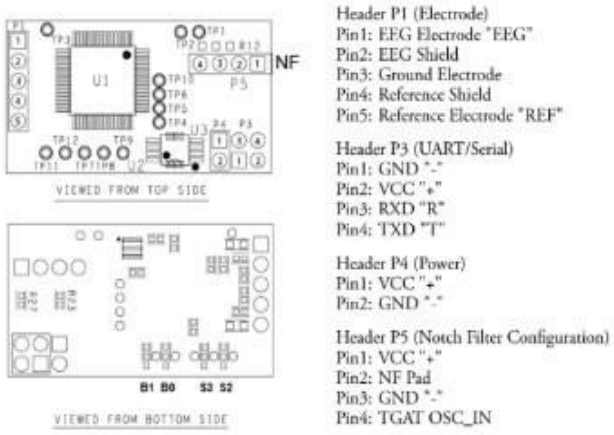

Figure 2: TGAM1 Top and bottom PIN diagram

- The dimension of the module is 29.9mm X 15.2mm X 2.5mm (1.1 in X 0.60 in X 0.10 in).
- Operating Voltage of this Module is 2.97 V – 3.63V.
- Maximum Input Noise, the module can filter is 10mV peak to peak. We will later measure our noise and will make sure that the noise is within the module range for optimum results.
- Maximum Power Consumption of the module is 15mA @3.3V. We will check these values with a multimeter and will measure each parameter.
- The device can communicate serially with 9600, 1200, 57600 bps baud rate. There are configuration pins with the help of which, we can change the baud rate.
- TGAM1 chip can handle only 1 EEG input, and we also need to process just one EEG channel.

2. Signal Processing and Analysis

The TGAM1 module processes raw EEG signals internally to provide meaningful data:

Filtering:

- A bandpass filter isolates frequencies between 0.5 Hz and 50 Hz, which correspond to brainwave activity, excluding low-frequency drift and high-frequency noise.

Brainwave Band Separation:

- Delta (0.5–4 Hz): Associated with deep sleep and relaxation.
- Theta (4–8 Hz): Related to creativity and meditation.
- Alpha (8–12 Hz): Reflects calmness and relaxation.
- Beta (12–30 Hz): Linked to concentration and alertness.
- Gamma (30–50 Hz): Involved in cognitive processing and learning.

FFT (Fast Fourier Transform):

- Internally or externally performed, FFT decomposes the EEG signal into its frequency components, enabling power spectrum analysis for brainwave bands.
- Artifact Removal:
- Algorithms identify and exclude artifacts (e.g., eye blinks, muscle contractions) to ensure clean data.

3. Feature Extraction

The TGAM1 processes the signal to extract features like:

- Attention Level: Indicates the user's focus or mental engagement.
- Meditation Level: Measures relaxation or mindfulness.
- Blink Detection: Identifies the frequency and intensity of blinks.

4. Output

The module outputs data in digital packets over a serial interface (e.g., Bluetooth or wired connection). Packets include:

Single-Byte CODEs:

Extended		(Byte)	
Code Level	[Code]	[Length]	Data Value Meaning
0	0x02	-	POOR-SIGNAL Quality (0-255)
0	0x03	-	HEART_RATE (0-255) Once/s on EGO
0	0x04	-	ATTENTION eSense (0-100)
0	0x05	-	MEDIATATION eSense (0-100)
0	0x06	-	8BIT_RAW Wave Value (0-255)
0	0x07	-	RAW_MARKER Section Start (0)

Multi-Byte CODEs:

Extended		(Byte)	

Code Level	[CODE]	[LENGTH]	Data Value Meaning
0	**0x80**	**2**	**RAW Wave Value: a single big-endian 16-bit two's-compliment signed value (high-order byte followed by low-order byte) (-32768 to 32767)**
0	0x81	32	EEG_POWER: eight big-endian 4-byte IEEE 754 floating point values representing delta, theta, low-alpha high-alpha, low-beta, high-beta, low-gamma, and mid-gamma EEG band power values
0	0x83	24	ASIC_EEG_POWER: eight big-endian 3-byte unsigned integer values representing delta, theta, low-alpha high-alpha, low-beta, high-beta, low-gamma, and mid-gamma EEG band power values
0	0x86	2	RRINTERVAL: two-byte big-endian unsigned integer representing the milliseconds between two R-peaks
Any	0x55	-	NEVER USED (reserved for [EXCODE])
Any	0xAA	-	NEVER USED (reserved for [SYNC])

***Note: The bold row is the output value we use to analyze frequency bands.**

II. Code explains in detail.

1. Recording real time EEG Signal with TGAM1.

a. Class initialization (__init__ method)

```
    def __init__(self, port, baudrate=57600):
        self.ser = serial.Serial(port, baudrate)
        self.data = {}
```

- **Purpose:** Initializes the ThinkGear object.
- **Parameters:**
  - port: The serial port to which the EEG device is connected (e.g., "COM3" or "/dev/ttyUSB0").
  - baudrate: Communication speed (defaults to 57600, typical for ThinkGear devices).
- **Attributes:**
  - self.ser: A serial.Serial object that handles communication with the device.
  - self.data: A dictionary to store parsed data received from the EEG device.

b. Data fetching (fetch_data method)

*** **Initialization**

```
    def fetch_data(self):
        self.data = {}
        while True:
```

```
            self.ser.read_until(b'\xaa\xaa')
```

- **Purpose**: Continuously tries to fetch valid packets.
- **Details**:
  - self.data = {}: Resets the data dictionary for fresh parsing.
  - self.ser.read_until(b'\xaa\xaa'): Reads from the serial stream until it detects the ThinkGear synchronization bytes (0xAA 0xAA), which mark the start of a data packet.

***** Reading the payload**

```
            payload = []
            checksum = 0
            packet_length = self.ser.read(1)
            payload_length = packet_length[0]
```

- **Purpose:** Reads the payload structure of the packet.
- **Details:**
  - payload = []: Initializes an empty list to store the payload data.
  - checksum = 0: Initializes the checksum value for validation.
  - packet_length = self.ser.read(1): Reads one byte representing the payload length.
  - payload_length = packet_length[0]: Converts the byte to an integer (length of the payload).

***** Extracting payload and validating checksum**

```
for i in range(payload_length):
                packet_code = self.ser.read(1)
                tempPacket = packet_code[0]
                payload.append(packet_code)
                checksum += tempPacket
            checksum = ~checksum & 0xff
            check = self.ser.read(1)
            if checksum == check[0]:
                break
            else:
                print('ERROR: Checksum mismatch!')
```

- **Purpose:** Reads the payload bytes, calculates the checksum, and validates it.

- **Details:**
    - Reading the Payload:
        - Loops through payload_length and reads each byte of the payload.
        - packet_code: Reads a single byte.
        - tempPacket: Extracts the integer value of the byte.
        - payload.append(packet_code): Appends the raw byte to the payload list.
        - checksum += tempPacket: Adds the byte value to the checksum.
    - Checksum Calculation:
        - checksum = ~checksum & 0xff: Computes the bitwise NOT of the checksum and ensures it's an 8-bit value.
    - Checksum Validation:
        - check = self.ser.read(1): Reads the checksum byte from the packet.
        - if checksum == check[0]: Compares the computed checksum with the received one.
        - If mismatched, prints an error and restarts the loop.

***** Parsing the payload**

```
i = 0
        while i < payload_length:
            packet = payload[i]
            if packet == b'\x80':  # EEG raw value
                i += 1
                i += 1
                val0 = payload[i]
                i += 1
                val1 = payload[i]
                raw_value = val0[0] * 256 + val1[0]
                if raw_value > 32768:
                    raw_value -= 65536
                self.data['eeg_raw'] = raw_value
            else:
                pass
            i += 1
```

- **Purpose:** Parses the payload to extract data fields.

- **Details:**
  - i = 0: Index for parsing the payload.
  - Loop: Iterates over the payload bytes:
    - packet = payload[i]: Reads the current byte.
    - if packet == b'\x80': Checks if the packet is of type 0x80, which corresponds to an EEG raw value in ThinkGear's protocol.
    - Parsing EEG Raw Value:
      - Skips two reserved bytes (i += 1 twice).
      - Reads two bytes (val0 and val1) representing the raw EEG value in big-endian format.
      - Combines the bytes:
        - raw_value = val0[0] * 256 + val1[0]: Combines the high and low bytes into a single 16-bit integer.
        - if raw_value > 32768: Converts it from unsigned to signed integer (two's complement representation).
        - self.data['eeg_raw'] = raw_value: Stores the parsed EEG value in self.data.
    - else: pass: Ignores other packet types for now.
    - i += 1: Moves to the next byte in the payload.

c. Closing the serial connection

```
    def close(self):
        self.ser.close()
```

- **Purpose:** Closes the serial connection when finished.
- **Details:**
  - self.ser.close(): Properly terminates communication with the EEG device.

2. Analyzing Raw EEG Signal with FFT to Brainwave Bands and graphing with Matplotlib.

a. Bandpass Filtering with SciPy

Purpose:

Raw EEG signals can have noise and unwanted frequency components. A bandpass filter is applied to retain only the frequencies of interest (e.g., 0.5 Hz to 50 Hz, corresponding to typical EEG brainwave bands).

How It Works:

The function bandpass_filter uses the following SciPy functions:

- scipy.signal.butter: Designs the filter coefficients (numerator b and denominator a) for the bandpass filter.
- scipy.signal.lfilter: Applies the designed filter to the raw EEG data.

Code in Detail:

```
def bandpass_filter(data, lowcut, highcut, fs, order=5):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return lfilter(b, a, data)
```

- butter(*order*, [low, high], *btype*='band') Designs a bandpass filter with a specified order and cutoff frequencies (lowcut and highcut).
- return lfilter(b, a, *data*) Applies the filter to the raw EEG signal.

b. Frequency Analysis with FFT

Purpose:

After filtering the raw EEG signal, the Fast Fourier Transform (FFT) is used to convert the time-domain signal into the frequency domain. This allows the power of specific frequency bands (Delta, Theta, etc.) to be analyzed.

How It Works:

The code uses the following functions:

- scipy.fft.fft: Computes the FFT of the signal to transform it from the time domain to the frequency domain.
- np.fft.fftfreq: Generates the corresponding frequencies for each FFT output.

Code in Detail:

```
fft_result = np.abs(fft(filtered_signal))[:BUFFER_SIZE // 2] / BUFFER_SIZE
freqs = np.fft.fftfreq(BUFFER_SIZE, d=1 / SAMPLING_RATE)[:BUFFER_SIZE // 2]
```

- fft(filtered_signal) Computes the FFT of the filtered signal.
- np.abs(fft(...)) Converts the complex FFT result to its magnitude (amplitude).
- BUFFER_SIZE // 2 Retains only the positive frequencies (FFT output is symmetric).
- np.fft.fftfreq(BUFFER_SIZE, *d*=1 / SAMPLING_RATE) Generates the frequency values corresponding to the FFT results.

c. Extracting Power for Brainwave Bands

Purpose:

After FFT, the power of each brainwave frequency band (Delta, Theta, etc.) is calculated. This power indicates the strength of the signal in each frequency range.

How It Works:

The code calculates the power for each frequency band:

```
power_bands = {band: 0 for band in FREQ_BANDS}
for band, (low, high) in FREQ_BANDS.items():
    indices = np.where((freqs >= low) & (freqs < high))
    power_bands[band] = np.sum(fft_result[indices] ** 2)
```

- np.where((freqs >= low) & (freqs < high)) Finds the indices of frequencies within the range of each band.
- fft_result[indices] ** 2 Squares the amplitude to compute the power for the corresponding frequencies.
- np.sum(...) Sums up the power for all frequencies within the band.

III. Summary

TGAM1 simplifies EEG processing by performing onboard acquisition, filtering, feature extraction, and outputting useful metrics, allowing users to develop applications without requiring extensive signal processing expertise. However, raw data access also enables advanced custom analysis if needed.

IV. Reference

1. NeuroSky. *ThinkGear Communications Protocol: Developer Documentation*. NeuroSky, 2017. PDF file.
2. Weiming. *Realtime Read Data from EEG Devices*. GitHub, 2023, https://github.com/weiming1122/realtime-read-data-from-EEG-devices.