Research and approaches

1

Identifying various biomedical signals and determining which movements they represent using Fourier transform involves analyzing the frequency components of the signals. Here's a guide on how to approach this problem:

Steps to Identify Biomedical Signals Using Fourier Transform

1. **Preprocessing the Signal:**
   - Filter the signal to remove noise.
   - Normalize the signal to have a consistent scale.
2. **Compute the Fourier Transform:**
   - Use a Fast Fourier Transform (FFT) to convert the time-domain signal into the frequency domain.
3. **Analyze the Frequency Spectrum:**
   - Identify the dominant frequencies and their amplitudes in the spectrum.
   - Compare the identified frequencies with known frequency ranges of different biomedical signals.

Common Biomedical Signals and Their Frequency Ranges

- **Electrocardiogram (ECG):**
  - **Frequency Range:** 0.05 Hz to 100 Hz
  - **Dominant Frequency Components:** P-wave, QRS complex, T-wave
  - **Movement Representation:** Cardiac cycles and heart rate
- **Electromyogram (EMG):**
  - **Frequency Range:** 20 Hz to 500 Hz
  - **Dominant Frequency Components:** Muscle contractions
  - **Movement Representation:** Muscle activity, force generation
- **Electroencephalogram (EEG):**
  - **Frequency Range:** 0.5 Hz to 50 Hz
  - **Dominant Frequency Bands:**
    - Delta (0.5-4 Hz): Deep sleep
    - Theta (4-8 Hz): Light sleep, relaxation
    - Alpha (8-13 Hz): Relaxed, awake state

        - Beta (13-30 Hz): Active thinking, alertness
        - Gamma (30-50 Hz): High-level information processing
    - **Movement Representation:** Brain activity, cognitive states
- **Mechanomyogram (MMG):**
    - **Frequency Range:** 5 Hz to 100 Hz
    - **Dominant Frequency Components:** Muscle vibrations
    - **Movement Representation:** Muscle activity, contractions

Implementation Example Using Python

Here's an example of how to implement this using Python:

python
Copy code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq

# Sample biomedical signal (replace with your actual signal data)
# Example: Simulated ECG signal
t = np.linspace(0, 10, 1000)
signal = 1.0 * np.sin(2 * np.pi * 1.0 * t)  # 1 Hz component
signal += 0.5 * np.sin(2 * np.pi * 60.0 * t)  # 60 Hz component (noise)

# Compute the Fourier Transform
N = len(signal)
T = t[1] - t[0]  # Sampling interval
yf = fft(signal)
xf = fftfreq(N, T)[:N//2]

# Plot the time-domain signal
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(t, signal)
plt.title('Time Domain Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# Plot the frequency spectrum
plt.subplot(2, 1, 2)
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
```

```
plt.title('Frequency Spectrum')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude')
plt.grid()
plt.show()

# Identify the dominant frequencies
dominant_freqs = xf[np.argsort(-np.abs(yf[:N//2]))][:5]
print("Dominant frequencies:", dominant_freqs)
```

Interpretation

- **Identify Dominant Frequencies:**
  - Look at the frequency spectrum plot and identify peaks.
  - Compare these peaks with the known frequency ranges of biomedical signals.
- **Match with Biomedical Signals:**
  - ECG: Peaks in the range of 0.05 to 100 Hz, typically around 1-2 Hz for heart rate.
  - EMG: Peaks in the range of 20 to 500 Hz.
  - EEG: Peaks in specific bands (delta, theta, alpha, beta, gamma).
  - MMG: Peaks in the range of 5 to 100 Hz.

Movement Representation

- **Cardiac Cycles (ECG):**
  - Dominant low-frequency components (1-2 Hz) indicate heart rate.
- **Muscle Activity (EMG):**
  - Higher frequency components (20-500 Hz) indicate muscle contractions.
- **Brain Activity (EEG):**
  - Different frequency bands indicate various cognitive and sleep states.
- **Muscle Vibrations (MMG):**
  - Frequencies in the 5-100 Hz range indicate muscle activity and contractions.

Conclusion

By analyzing the frequency spectrum of a biomedical signal using Fourier transform, you can identify the dominant frequencies and match them with known frequency ranges of different biomedical signals. This helps in understanding the underlying physiological activities and movements represented by the signals.

2

Identifying Eye Movements

Blink Detection:

- **Characteristics:**
  - Blinks typically produce large, slow deflections in the EOG signal.
  - In the time domain, they look like large spikes.
  - They appear as low-frequency components in the frequency domain.
- **Time Domain Analysis:**
  - Look for sudden, significant peaks or spikes in the EOG signal.
  - The amplitude of these spikes is usually much higher than normal eye movement signals.
- **Frequency Domain Analysis:**
  - Blinks contribute primarily to the low-frequency range.
  - Use a low-pass filter to isolate the blink signals.

Example:

```
# Assuming `eog_signal` contains the raw EOG data
blink_spikes = eog_signal[eog_signal > threshold]  # threshold to be determined based on data
```

Saccades (Rapid Eye Movements):

- **Characteristics:**
  - Saccades generate sharp, high-frequency components in the EOG signal.
  - These movements show up as bursts in the higher frequency range of the spectrum.
- **Time Domain Analysis:**
  - Detect rapid changes or sharp transitions in the signal.
  - Saccades are characterized by quick, high-amplitude changes.
- **Frequency Domain Analysis:**
  - Saccades contribute to higher frequencies.
  - Use a high-pass filter to isolate saccadic movements.

Example:

```
# Identify sharp transitions
saccade_indices = np.where(np.abs(np.diff(eog_signal)) > threshold)  # threshold based on data
```

Smooth Pursuit:

- **Characteristics:**
    - Smooth, continuous movements of the eyes following a target.
    - These will have lower frequency components compared to saccades and less sharp transitions.
- **Time Domain Analysis:**
    - Identify segments of the signal with smooth, gradual changes.
    - The amplitude changes are more moderate compared to saccades.
- **Frequency Domain Analysis:**
    - These movements contribute to the mid-frequency range.
    - Use a band-pass filter to isolate smooth pursuit movements.

Example:

```
# Filter for smooth pursuit frequency range
smooth_pursuit_signal = bandpass_filter(eog_signal, low_freq, high_freq, fs, order=4)
```

Fixations:

- **Characteristics:**
    - When the eyes are stationary, the EOG signal shows minimal activity.
    - Dominant frequencies will be very low, close to DC (0 Hz).
- **Time Domain Analysis**:
    - Look for periods with little to no change in the signal.
    - Fixations are characterized by flat, stable segments.
- **Frequency Domain Analysis:**
    - Fixations contribute primarily to the very low-frequency range.
    - Use a low-pass filter to detect fixations.

Example:

```
# Identify periods of minimal activity
fixation_indices = np.where(np.abs(np.diff(eog_signal)) < threshold)  # threshold based on data
```

Pattern Recognition

Feature Extraction:

- **Time Domain Features:**
  - Amplitude: Measure the peak and trough values.
  - Duration: Measure the length of time the signal stays in a particular state.
  - Spike Count: Number of rapid transitions or spikes.
- **Frequency Domain Features:**
  - **Dominant Frequencies:** Identify peaks in the frequency spectrum.
  - **Power Spectral Density (PSD):** Measure the power distribution across frequencies.

Example:

```
# Extracting features
amplitude = np.max(eog_signal) - np.min(eog_signal)
duration = len(eog_signal)
spike_count = len(np.where(np.abs(np.diff(eog_signal)) > spike_threshold)[0])
```

-

Classification:

- Model Selection:
  - Use labeled data to train a classifier.
  - Common models include Support Vector Machine (SVM), Random Forest, and Neural Networks.
- Feature Engineering:
  - Combine statistical measures, frequency components, and time-domain characteristics.
  - Normalize and scale features as necessary.

Example:

```
from sklearn.ensemble import RandomForestClassifier

# Assuming X_train contains features and y_train contains labels
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
```

Training and Validation:

- **Dataset Split:**
  - Split your dataset into training and testing sets (e.g., 70% training, 30% testing).
- **Model Training:**

- Train your model on the training set.
- **Model Validation:**
  - Validate the model on the testing set to ensure it generalizes well.
  - Use metrics like accuracy, precision, recall, and F1-score to evaluate performance.

Example:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model
classifier.fit(X_train, y_train)

# Validate the model
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

By following these detailed steps, you can systematically identify and classify various eye movements from EOG signals, even if you are starting your research and do not have prior knowledge of the exact frequency ranges.

RESULT-3 (Processing for signal and blink detection)

To distinguish between your EOG signal and noise, and to keep signals of only a certain amplitude, you can use several signal processing techniques. Here's an outline of the steps you can follow:

1. **Bandpass Filtering**: This will help isolate the frequency range of interest and remove high-frequency noise and low-frequency drift.
2. **Amplitude Thresholding**: After filtering, apply amplitude thresholding to identify and keep signals within a certain amplitude range.
3. **Peak Detection**: Identify peaks in the signal that correspond to eye movements (blinks).

Below is the code that incorporates these steps. This code will:

- Read the EOG signal from a CSV file.

- Apply a bandpass filter to remove noise.
- Apply amplitude thresholding to keep signals of certain amplitude.
- Detect blinks by finding peaks in the thresholded signal.

python
Copy code
```
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, ifft, fftfreq
from scipy.signal import butter, lfilter, find_peaks

# Function to design a bandpass filter
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return b, a

# Function to apply bandpass filter to data
def bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

# File to read data from
blink_data = 'eog_data_blink.csv'

# Initialize lists to store data
timestamps = []
voltages = []

# Read data from the CSV file for plotting
with open(blink_data, mode='r') as file: # Open the file for reading
    reader = csv.reader(file) # Create a CSV reader object
    next(reader)  # Skip the header row
    for row in reader:
        timestamps.append(float(row[0]))
        voltages.append(float(row[1]))
```

```
# Convert lists to numpy arrays for processing
timestamps = np.array(timestamps)
voltages = np.array(voltages)

# Sampling rate (assuming timestamps are in milliseconds)
sampling_rate = 1000 / (timestamps[1] - timestamps[0])

# Apply bandpass filter to remove noise
lowcut = 0.1  # Low cut-off frequency in Hz
highcut = 30.0  # High cut-off frequency in Hz
filtered_signal = bandpass_filter(voltages, lowcut, highcut, sampling_rate, order=6)

# Compute the Fourier Transform of the filtered signal
N = len(filtered_signal)
T = 1.0 / sampling_rate
yf = fft(filtered_signal)
xf = fftfreq(N, T)[:N//2]

# Plot the original and filtered time-domain signal
plt.figure(figsize=(12, 8))
plt.subplot(3, 1, 1)
plt.plot(timestamps, voltages)
plt.title('Original Time Domain Signal')
plt.xlabel('Time [ms]')
plt.ylabel('Voltage (V)')

plt.subplot(3, 1, 2)
plt.plot(timestamps, filtered_signal)
plt.title('Filtered Time Domain Signal')
plt.xlabel('Time [ms]')
plt.ylabel('Voltage (V)')

# Plot the frequency spectrum of the filtered signal
plt.subplot(3, 1, 3)
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.title('Frequency Spectrum of Filtered Signal')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude')
plt.grid()
plt.tight_layout()
```

```
plt.show()

# Amplitude thresholding to keep signals of certain amplitude
amplitude_threshold = np.mean(filtered_signal) + 2 * np.std(filtered_signal)
thresholded_signal = np.where(np.abs(filtered_signal) > amplitude_threshold, filtered_signal, 0)

# Blink detection using peak finding
peak_indices, _ = find_peaks(thresholded_signal, height=amplitude_threshold)
blink_times = timestamps[peak_indices]

# Print and save blink times
print("Blink times:", blink_times)

# Save blink times to a file
with open('blink_times.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Blink Time (ms)'])
    for blink_time in blink_times:
        writer.writerow([blink_time])

# Save filtered signal to a file
with open('filtered_signal.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Time (ms)', 'Filtered Voltage (V)'])
    for time, voltage in zip(timestamps, filtered_signal):
        writer.writerow([time, voltage])

# Plot the thresholded signal with detected blinks
plt.figure(figsize=(12, 6))
plt.plot(timestamps, thresholded_signal, label='Thresholded Signal')
plt.plot(timestamps[peak_indices], thresholded_signal[peak_indices], 'rx', label='Detected
Blinks')
plt.title('Thresholded Signal with Detected Blinks')
plt.xlabel('Time [ms]')
plt.ylabel('Voltage (V)')
plt.legend()
plt.show()
```

Explanation

1. **Bandpass Filtering**:
   - butter_bandpass and bandpass_filter functions design and apply a bandpass filter to the signal.
   - The lowcut and highcut frequencies are set to 0.1 Hz and 30 Hz, respectively, to remove low-frequency drift and high-frequency noise.
2. **Fourier Transform**:
   - The Fourier Transform of the filtered signal is computed and plotted to visualize the frequency components.
3. **Amplitude Thresholding**:
   - The threshold is calculated as the mean plus two standard deviations of the filtered signal.
   - Signals exceeding this threshold are kept, and others are set to zero.
4. **Blink Detection**:
   - The find_peaks function from scipy.signal is used to detect peaks in the thresholded signal, which correspond to blinks.
   - The blink times are extracted and saved to a CSV file.
5. **Plotting**:
   - The original signal, filtered signal, frequency spectrum, and thresholded signal with detected blinks are plotted for visualization.

By adjusting the lowcut, highcut, and amplitude_threshold values, you can refine the signal processing to better isolate the EOG signal from noise and accurately detect blinks.

4 -

how to know which peak in frequency domain represents what? signal or noise
ChatGPT

Identifying which peaks in the frequency domain represent signal and which represent noise can be challenging, but there are several techniques and principles that can help you make this distinction. Here are some common methods:

1. Domain Knowledge

- **Prior Knowledge**: Knowing the expected frequency range of the signal can help distinguish between signal and noise. For instance, in biomedical signals like ECG, EOG, or EEG, certain frequency ranges are well-documented.
- **Expected Harmonics**: Signals often have harmonics (multiples of the fundamental frequency). Recognizing these patterns can help differentiate signals from random noise.

2. Amplitude

- **Significant Peaks**: True signals usually have significant peaks compared to the background noise. By setting a threshold, you can filter out low-amplitude noise.
- **Noise Floor**: The noise floor represents the average level of background noise. Peaks significantly above this floor are more likely to be true signals.

3. Filtering Techniques

- **Bandpass Filters**: Applying a bandpass filter that matches the expected frequency range of the signal can help isolate the signal from noise.
- **Low-Pass and High-Pass Filters**: If you know the signal is below or above a certain frequency, these filters can remove unwanted components.

4. Frequency Domain Analysis

- **Fourier Transform**: Use the Fast Fourier Transform (FFT) to convert the time-domain signal into the frequency domain. Peaks in the frequency domain correspond to periodic components of the signal.
- **Spectral Analysis**: Analyze the power spectral density (PSD) to understand the distribution of power across frequencies.

5. Statistical Methods

- **Signal-to-Noise Ratio (SNR)**: Calculate the SNR to quantify the level of the signal relative to the noise.
- **Thresholding**: Use statistical measures to set thresholds. Peaks above a certain number of standard deviations from the mean can be considered signals.

Example Code

Here’s an example of how you can analyze an EOG signal to identify signal and noise peaks, including filtering and thresholding techniques:

python
Copy code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import butter, lfilter

# Example EOG data (replace with actual data)
sampling_rate = 1000  # Hz
t = np.linspace(0, 10, sampling_rate * 10)
eog_signal = 1.0 * np.sin(2 * np.pi * 1.0 * t)  # 1 Hz component (signal)
eog_signal += 0.5 * np.sin(2 * np.pi * 60.0 * t)  # 60 Hz component (noise)
eog_signal += 0.5 * np.random.randn(len(t))  # Random noise

# Bandpass Filter Design
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return b, a

def bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

# Apply Bandpass Filter
lowcut = 0.1  # Hz
highcut = 30.0  # Hz
filtered_signal = bandpass_filter(eog_signal, lowcut, highcut, sampling_rate, order=6)

# Compute Fourier Transform
N = len(filtered_signal)
T = 1.0 / sampling_rate
yf = fft(filtered_signal)
xf = fftfreq(N, T)[:N//2]

# Plot Time-Domain Signal
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(t, filtered_signal)
```

```
plt.title('Filtered Time Domain Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

# Plot Frequency Spectrum
plt.subplot(2, 1, 2)
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.title('Frequency Spectrum')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude')
plt.grid()
plt.show()

# Identify Dominant Frequencies
dominant_freqs = xf[np.argsort(-np.abs(yf[:N//2]))][:5]
print("Dominant frequencies:", dominant_freqs)

# Threshold to Identify Peaks (Example: 0.1 * max amplitude)
threshold = 0.1 * max(np.abs(yf[:N//2]))
signal_peaks = xf[np.abs(yf[:N//2]) > threshold]
print("Signal Peaks:", signal_peaks)

# Apply Inverse Fourier Transform to remove noise
yf[np.abs(yf) < threshold] = 0
cleaned_signal = np.fft.ifft(yf)

# Plot Cleaned Time-Domain Signal
plt.figure(figsize=(12, 6))
plt.plot(t, cleaned_signal.real)
plt.title('Cleaned Time Domain Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.show()
```

Explanation

1. **Filtering**: A bandpass filter is used to retain frequencies within a specified range, removing high-frequency noise.
2. **Fourier Transform**: The FFT converts the signal to the frequency domain, showing the frequency components.

3. **Peak Identification**: Dominant frequencies are identified based on amplitude.
4. **Thresholding**: Peaks above a certain amplitude threshold are considered signals.
5. **Inverse Fourier Transform**: Used to reconstruct the time-domain signal after filtering out noise frequencies.

Practical Considerations

- **Frequency Range**: Adjust the filter's cut-off frequencies to match the expected signal range.
- **Threshold Setting**: Choose a threshold based on the signal-to-noise ratio and specific application requirements.
- **Multiple Peaks**: Signals may have multiple harmonics; ensure they are correctly identified.

Using these techniques, you can analyze and clean EOG (or any other biomedical) signals to differentiate between useful data and noise.