Sanjida Israt Suchona

Roll:220611

implementing a complete PPG signal processing pipeline, including filtering, peak and valley detection, and heart rate estimation

Introduction:

Photoplethysmography (PPG) is a non-invasive optical technique used to measure blood volume changes in the microvascular tissue.It is commonly used in wearable devices and medical applications to monitor heart rate and oxygen saturation. The PPG signal consists of pulsatile components corresponding to the heartbeat and is often contaminated by noise, requiring signal processing techniques to extract useful information.

Objectives:

1. Generate a simulated PPG signal with added noise to mimic real-world conditions.
2. Apply a Butterworth low-pass filter to remove high-frequency noise and smooth the signal.
3. Detect peaks and valleys in the filtered PPG signal to identify heartbeats.
4. Identify abnormal peaks and valleys based on predefined amplitude thresholds.
5. Calculate heart rate based on detected peaks and valleys.
6. Classify heart rate conditions as normal, low, or high based on standard thresholds.
7. Visualize the PPG signal with detected peaks, valleys, and abnormalities for analysis.

Procedure:

1. Initialize Parameters
   - Set the sampling frequency to 100 Hz.
   - Generate a synthetic PPG signal using a sine wave with added noise.

2. Filter the PPG Signal
   - Design a Butterworth low-pass filter with a cutoff frequency of 3 Hz.
   - Apply the filter to remove high-frequency noise from the signal.

3. Detect Peaks and Valleys
   - Use the find_peaks() function to locate systolic peaks (high points).
   - Identify diastolic valleys (low points) in the filtered signal.

4. Calculate Heart Rate (BPM)
   - Compute time intervals between consecutive systolic peaks.
   - Convert the intervals into beats per minute (BPM).

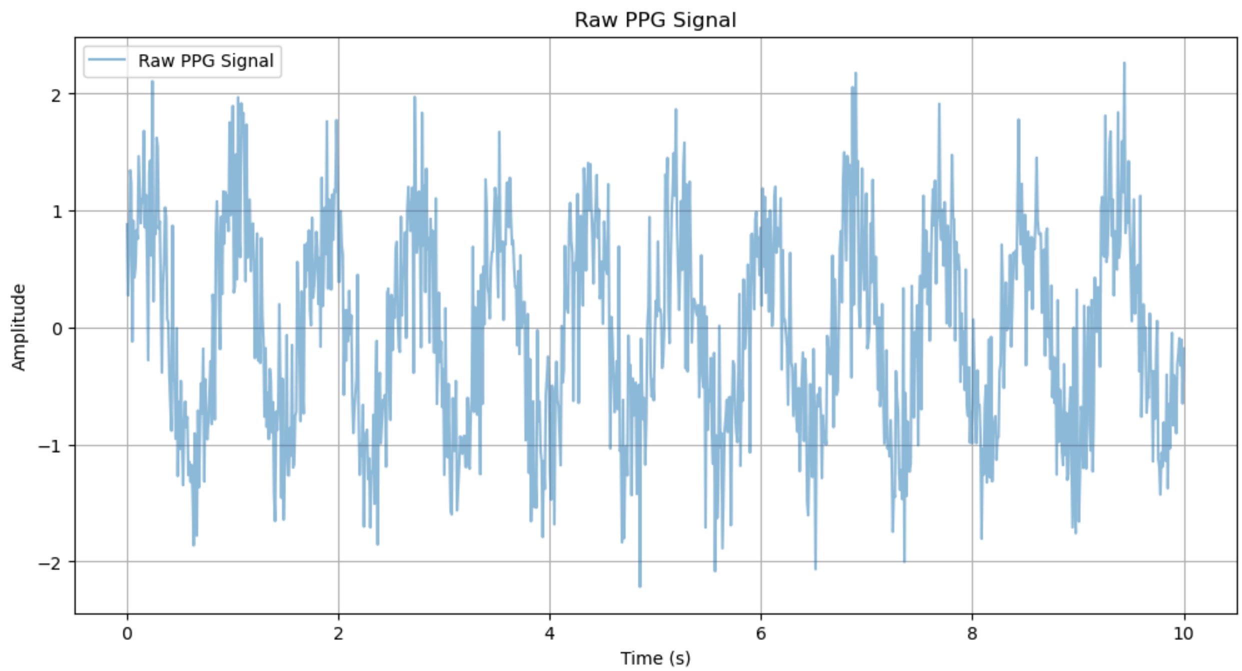
5. Classify Heart Rate
   - If BPM is between 50-100, classify as normal.
   - If BPM is greater than 100, classify as high.
   - If BPM is less than 50, classify as low.

6. Detect Abnormalities
   - Identify abnormal peaks or valleys by finding outliers.
   - Adjust heart rate calculation based on detected abnormalities.

7. Plot the Results
   - Display the filtered PPG signal.
   - Mark detected peaks and valleys.
   - Highlight abnormal points for visualization.

Generate sample ppg signal and define the low-pass filter ,apply low-pass filter ,detect peak and then plot Raw ppg signal:

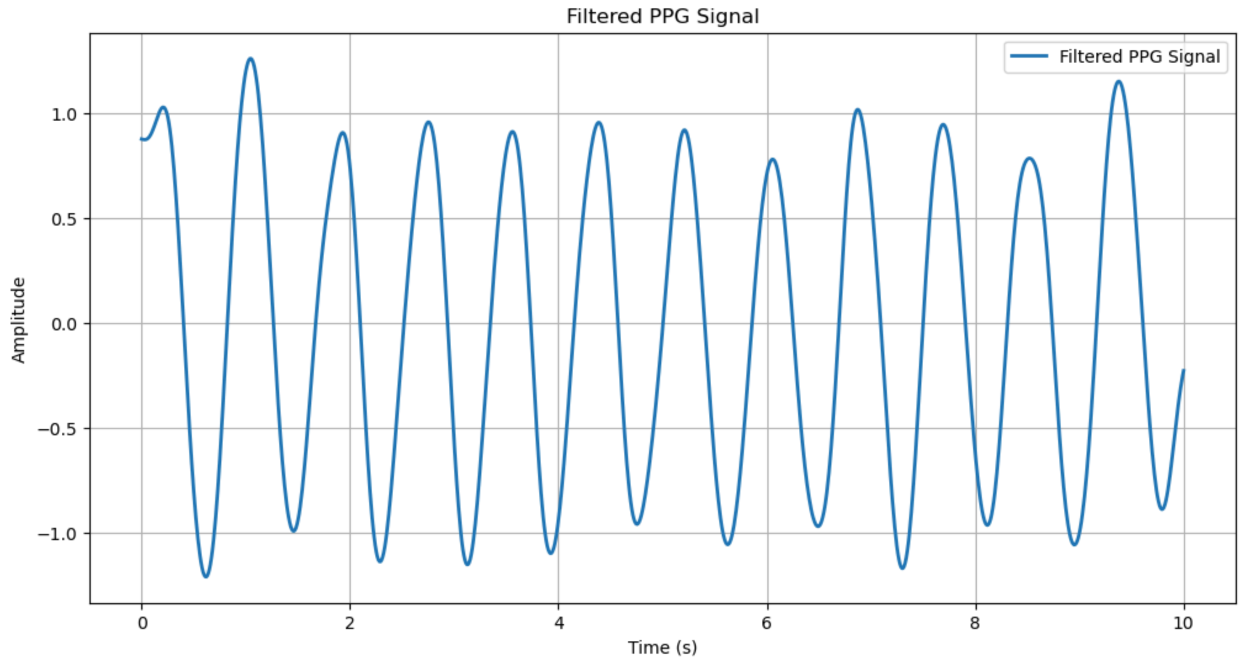
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

# Generate sample PPG signal
np.random.seed(0)
time = np.linspace(0, 10, 1000)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))  # Fixed

# Define the lowpass filter function
def butter_lowpass_filter(data, cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a, data)
    return y  # Return the filtered signal

# Apply Lowpass Filter
fs = 100  # Sampling frequency (assuming 100 Hz)
cutoff = 3  # Cutoff frequency for the filter
filtered_signal = butter_lowpass_filter(ppg_signal, cutoff, fs)

# Detect peaks (heartbeats)
peaks, _ = find_peaks(filtered_signal, height=0)  # Detect peaks above 0

# Plot 1: Raw PPG Signal
plt.figure(figsize=(12, 6))
plt.plot(time, ppg_signal, label="Raw PPG Signal", alpha=0.5)
plt.title("Raw PPG Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```



Generate sample ppg signal, filter settings and then plot filtered ppg signal:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

# Generate sample PPG signal
np.random.seed(0)
time = np.linspace(0, 10, 1000)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))

# Filter settings
fs = 100  # Sampling frequency in Hz
cutoff = 3  # Cutoff frequency in Hz
filtered_ppg = butter_lowpass_filter(ppg_signal, cutoff, fs)
#  Plot 2: Filtered PPG Signal
plt.figure(figsize=(12, 6))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
plt.title("Filtered PPG Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
```

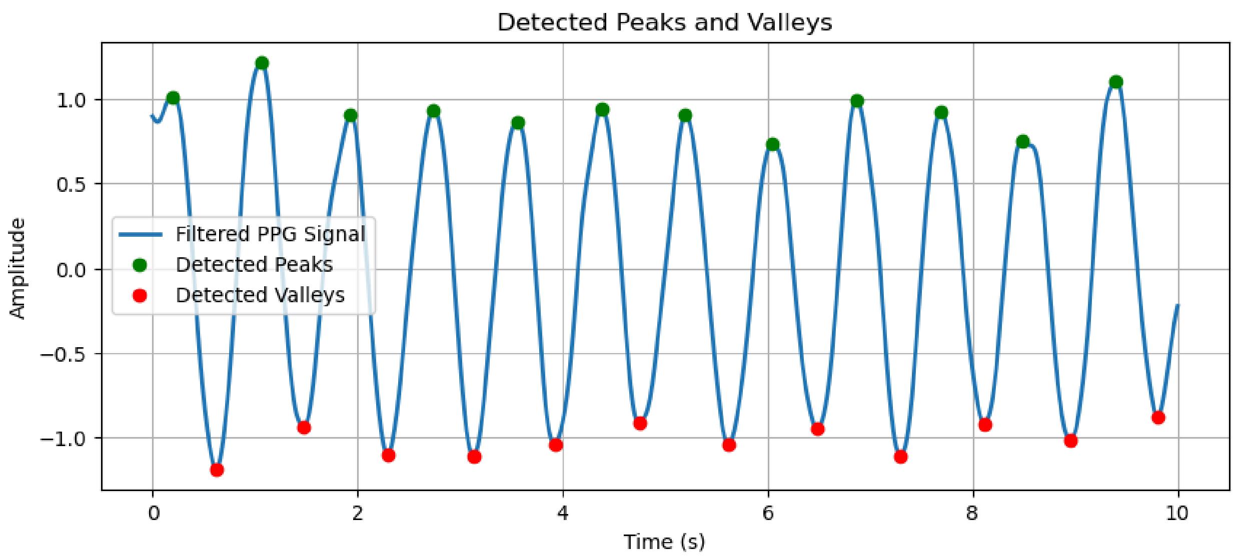
```
plt.legend()
plt.show()
```



Detected peak and valleys and plot peaks and valley:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

np.random.seed(0)
time = np.linspace(0, 10, 1000)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))
# Peak and valley detection
peaks, _ = find_peaks(filtered_ppg, height=0.5, distance=fs//2)
valleys, _ = find_peaks(-filtered_ppg, height=0.5, distance=fs//2)
# Plot 3: Peaks and Valleys
plt.figure(figsize=(10, 4))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
```

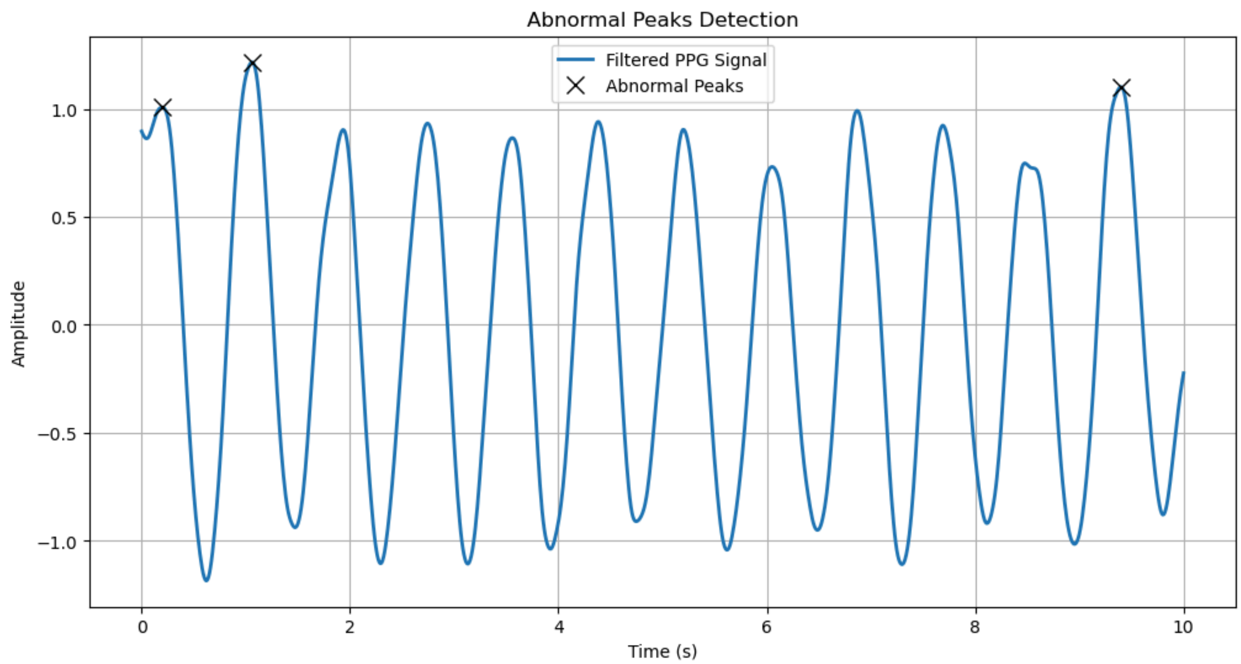
```
plt.plot(time[peaks], filtered_ppg[peaks], "go", label="Detected Peaks")
plt.plot(time[valleys], filtered_ppg[valleys], "ro", label="Detected
Valleys")
plt.title("Detected Peaks and Valleys")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```



Abnormal peaks detection and plot abnormal peaks:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

np.random.seed(0)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))
# Abnormal peaks detection
peak_heights = filtered_ppg[peaks]
abnormal_peaks = peaks[peak_heights > 1]  # Threshold for high spikes
```

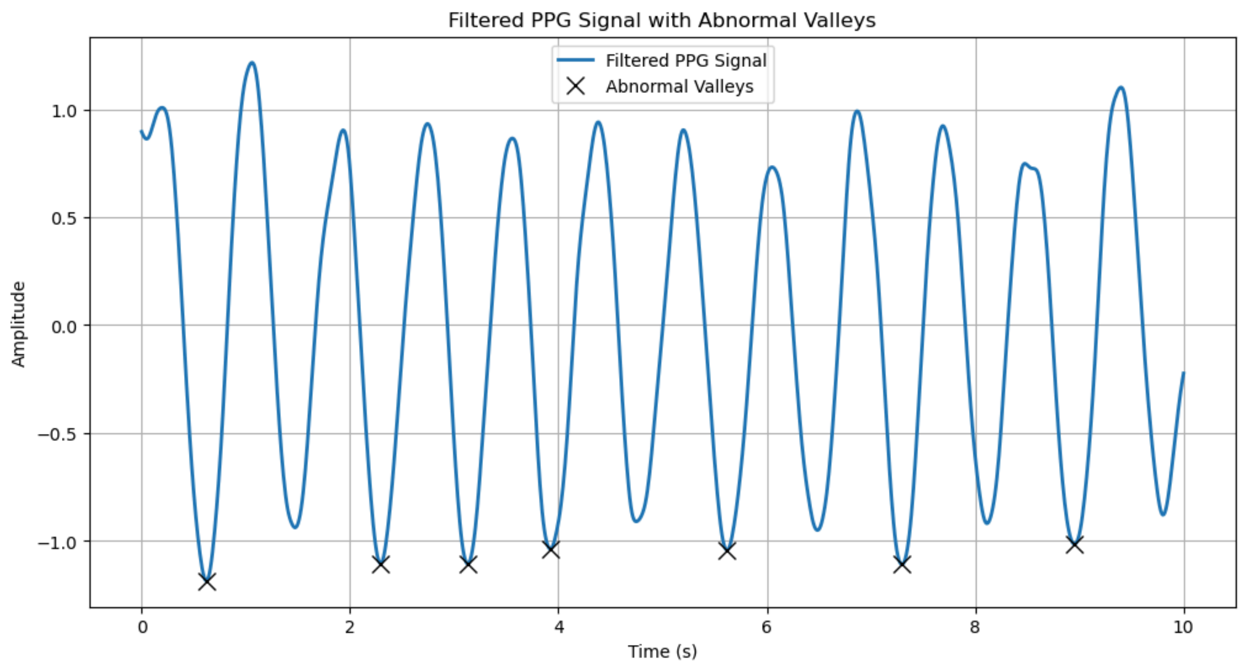
```
# Plot 4: Abnormal Peaks
plt.figure(figsize=(12, 6))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
plt.plot(time[abnormal_peaks], filtered_ppg[abnormal_peaks], "kx",
label="Abnormal Peaks", markersize=10)
plt.title("Abnormal Peaks Detection")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```



Simulated time array,sampling frequency, simulated ppg signal with noise,apply butterworth low-pass filter,detect valleys ,define abnormal valleys and plot filtered signal with abnormal valleys:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

# Simulated time array
```

```python
fs = 100  # Sampling frequency in Hz
duration = 10  # seconds
time = np.linspace(0, duration, fs * duration)

# Simulated PPG signal with noise
np.random.seed(0)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))

# Apply Butterworth Low-Pass Filter
def butter_lowpass_filter(data, cutoff=3.0, fs=100, order=2):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

filtered_ppg = butter_lowpass_filter(ppg_signal)

# Detect Valleys (Negative Peaks)
valleys, _ = find_peaks(-filtered_ppg)

# Define Abnormal Valleys (Threshold-Based)
valley_depths = filtered_ppg[valleys]
abnormal_valleys = valleys[valley_depths < -1.0]  # Adjust threshold as
needed

# Plot Filtered Signal with Abnormal Valleys
plt.figure(figsize=(12, 6))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
plt.plot(time[abnormal_valleys], filtered_ppg[abnormal_valleys], "kx",
label="Abnormal Valleys", markersize=10)

plt.title("Filtered PPG Signal with Abnormal Valleys")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```

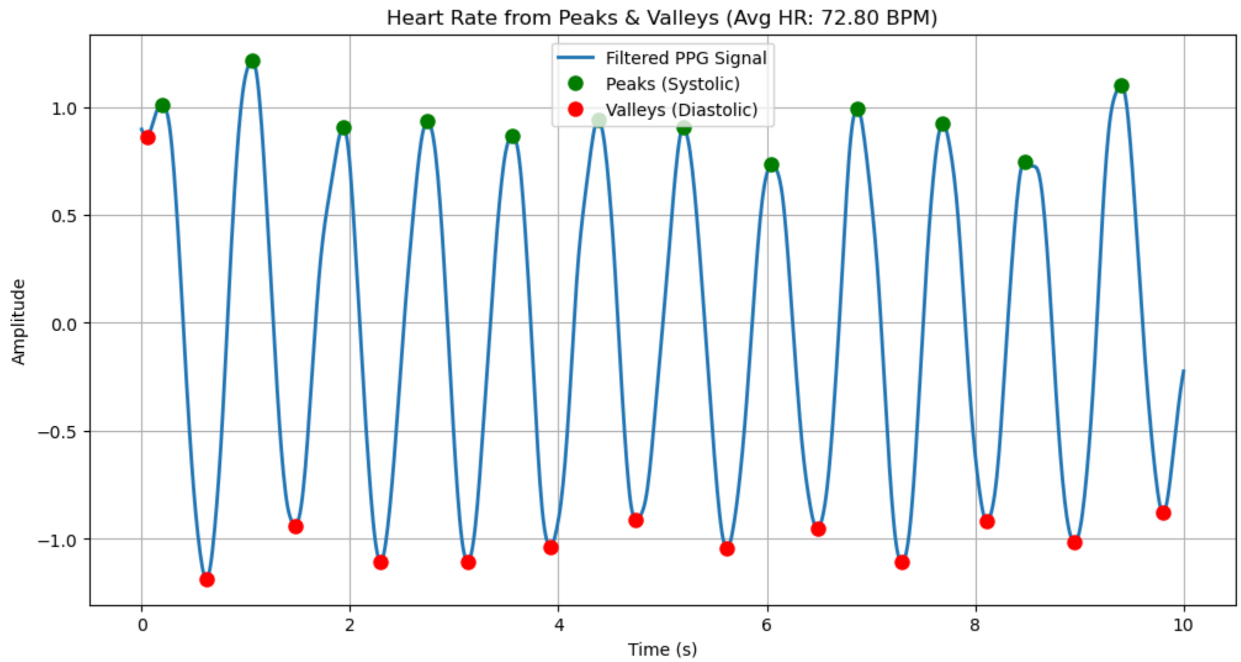


Simulated time array,sampling frequency, simulated ppg signal with noise,apply butterworth low-pass filter,detect peaks and valleys, calculate heart rate from peaks and valleys,final heart rate and plot ppg signal with peaks and valleys.Then we give a condition that is;
If heart rate greater than 100 then high heart rate else if heart rate less than 50 low heart rate or normal heart rate and we get normal heart rate.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

# Simulated time array
fs = 100  # Sampling frequency in Hz
duration = 10  # Total duration in seconds
time = np.linspace(0, duration, fs * duration)

# Simulated PPG signal with noise
np.random.seed(0)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))

# Apply Butterworth Low-Pass Filter
def butter_lowpass_filter(data, cutoff=3.0, fs=100, order=2):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

filtered_ppg = butter_lowpass_filter(ppg_signal)

# **Detect Peaks (Systolic)**
peaks, _ = find_peaks(filtered_ppg, distance=fs//2)  # Ensure at least
0.5s gap

# **Detect Valleys (Diastolic)**
valleys, _ = find_peaks(-filtered_ppg, distance=fs//2)  # Inverted signal
to detect valleys

# **Calculate Heart Rate from Peaks**
if len(peaks) > 1:
    peak_intervals = np.diff(time[peaks])  # Time intervals between peaks
    heart_rate_peaks = 60 / np.mean(peak_intervals)  # Convert to BPM
else:
    heart_rate_peaks = 0

# **Calculate Heart Rate from Valleys**
if len(valleys) > 1:
```

```python
    valley_intervals = np.diff(time[valleys])  # Time intervals between valleys
    heart_rate_valleys = 60 / np.mean(valley_intervals)  # Convert to BPM
else:
    heart_rate_valleys = 0

# **Final Heart Rate (Average of Both Methods)**
heart_rate_avg = (heart_rate_peaks + heart_rate_valleys) / 2 if heart_rate_peaks and heart_rate_valleys else max(heart_rate_peaks, heart_rate_valleys)

# **Plot PPG Signal with Peaks and Valleys**
plt.figure(figsize=(12, 6))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
plt.plot(time[peaks], filtered_ppg[peaks], "go", label="Peaks (Systolic)", markersize=8)
plt.plot(time[valleys], filtered_ppg[valleys], "ro", label="Valleys (Diastolic)", markersize=8)

plt.title(f"Heart Rate from Peaks & Valleys (Avg HR: {heart_rate_avg:.2f} BPM)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()

print(f"Heart Rate from Peaks: {heart_rate_peaks:.2f} BPM")
print(f"Heart Rate from Valleys: {heart_rate_valleys:.2f} BPM")
print(f"Final Estimated Heart Rate: {heart_rate_avg:.2f} BPM")
if heart_rate_avg > 100:
    print("⚠️ Warning: High Heart Rate Detected!")
elif heart_rate_avg < 50:
    print("⚠️ Warning: Low Heart Rate Detected!")
else:
    print("✅ Normal Heart Rate.")
```



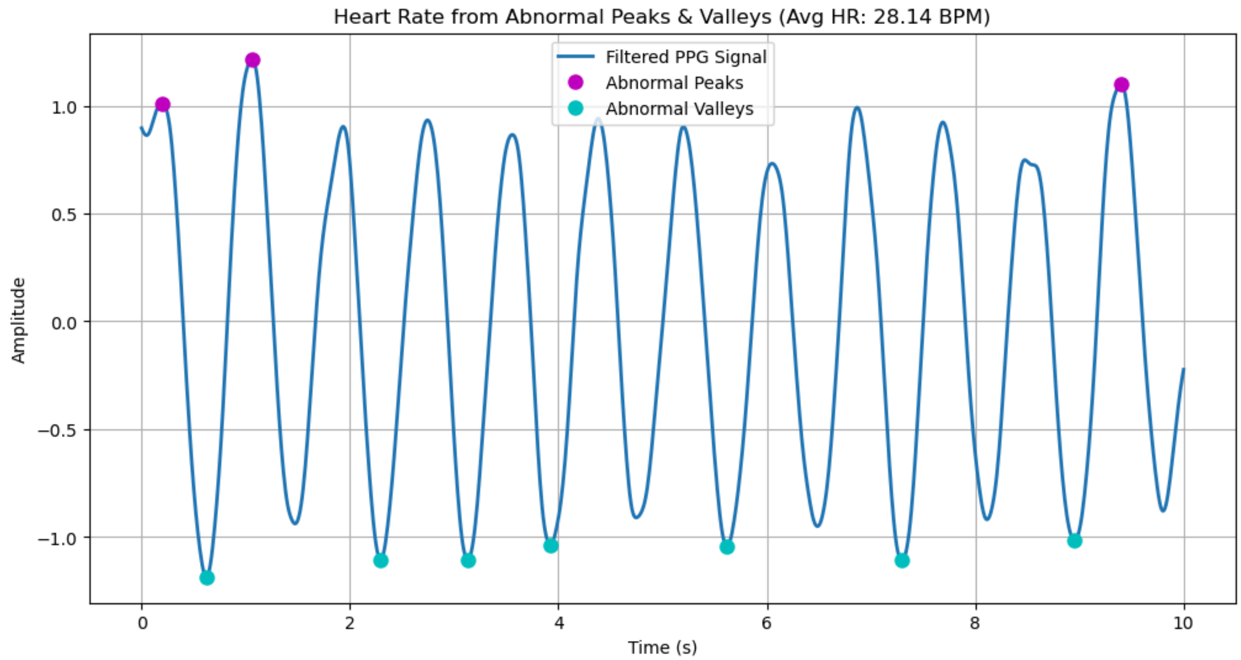

```
Heart Rate from Peaks: 71.75 BPM
Heart Rate from Valleys: 73.85 BPM
Final Estimated Heart Rate: 72.80 BPM
✅ Normal Heart Rate.
```

Simulated time array,sampling frequency, simulated ppg signal with noise,apply butterworth low-pass filter,detect peaks and valleys, calculate heart rate from abnormal peak and valleys,final heart rate and plot ppg signal with abnormal peaks and valleys.Then we give a condition that is;
If heart rate greater than 100 then high heart rate else if heart rate less than 50 low heart rate or normal heart rate and we get low heart rate.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt

# Simulated time array
```

```python
fs = 100  # Sampling frequency in Hz
duration = 10  # Total duration in seconds
time = np.linspace(0, duration, fs * duration)

# Simulated PPG signal with noise
np.random.seed(0)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))

# Apply Butterworth Low-Pass Filter
def butter_lowpass_filter(data, cutoff=3.0, fs=100, order=2):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

filtered_ppg = butter_lowpass_filter(ppg_signal)

# **Detect Peaks (Systolic)**
peaks, _ = find_peaks(filtered_ppg, distance=fs//2)

# **Detect Valleys (Diastolic)**
valleys, _ = find_peaks(-filtered_ppg, distance=fs//2)

# **Define Abnormal Peaks (High Spikes)**
peak_heights = filtered_ppg[peaks]
abnormal_peaks = peaks[peak_heights > 1.0]  # Adjust threshold as needed

# **Define Abnormal Valleys (Deep Drops)**
valley_depths = filtered_ppg[valleys]
abnormal_valleys = valleys[valley_depths < -1.0]  # Adjust threshold as
needed

# **Calculate Heart Rate from Abnormal Peaks**
if len(abnormal_peaks) > 1:
    peak_intervals = np.diff(time[abnormal_peaks])
    heart_rate_peaks = 60 / np.mean(peak_intervals)
else:
    heart_rate_peaks = 0
```

```python
# **Calculate Heart Rate from Abnormal Valleys**
if len(abnormal_valleys) > 1:
    valley_intervals = np.diff(time[abnormal_valleys])
    heart_rate_valleys = 60 / np.mean(valley_intervals)
else:
    heart_rate_valleys = 0

# **Final Heart Rate (Average of Both Methods)**
heart_rate_avg = (heart_rate_peaks + heart_rate_valleys) / 2 if
heart_rate_peaks and heart_rate_valleys else max(heart_rate_peaks,
heart_rate_valleys)

# **Plot PPG Signal with Abnormal Peaks and Valleys**
plt.figure(figsize=(12, 6))
plt.plot(time, filtered_ppg, label="Filtered PPG Signal", linewidth=2)
plt.plot(time[abnormal_peaks], filtered_ppg[abnormal_peaks], "mo",
label="Abnormal Peaks", markersize=8)
plt.plot(time[abnormal_valleys], filtered_ppg[abnormal_valleys], "co",
label="Abnormal Valleys", markersize=8)

plt.title(f"Heart Rate from Abnormal Peaks & Valleys (Avg HR:
{heart_rate_avg:.2f} BPM)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()

print(f"Heart Rate from Abnormal Peaks: {heart_rate_peaks:.2f} BPM")
print(f"Heart Rate from Abnormal Valleys: {heart_rate_valleys:.2f} BPM")
print(f"Final Estimated Heart Rate: {heart_rate_avg:.2f} BPM")
if heart_rate_avg > 100:
    print("⚠️ Warning: High Heart Rate Detected!")
elif heart_rate_avg < 50:
    print("⚠️ Warning: Low Heart Rate Detected!")
else:
    print("✅ Normal Heart Rate.")
```




```
Heart Rate from Abnormal Peaks: 13.04 BPM
Heart Rate from Abnormal Valleys: 43.23 BPM
Final Estimated Heart Rate: 28.14 BPM
⚠️ Warning: Low Heart Rate Detected!
```

Simulated ECG signal,detect R-peaks in ECG,compare ECG and PPG heart rates:

```
from scipy.signal import find_peaks

# Simulated ECG Signal (Replace with real sensor data)
ecg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.3 *
np.random.normal(size=len(time))

# Detect R-Peaks in ECG
ecg_peaks, _ = find_peaks(ecg_signal, height=0.5, distance=50)

# Compare ECG and PPG Heart Rates
```

```python
ecg_hr = 60 / np.mean(np.diff(time[ecg_peaks]))  # ECG-based HR
ppg_hr = 60 / np.mean(np.diff(time[peaks]))  # PPG-based HR

if abs(ecg_hr - ppg_hr) > 10:  # Threshold difference
    print("⚠️ Possible Motion Artifact: ECG & PPG Heart Rates Do Not
Match!")
else:
    print(f"✅ ECG HR: {ecg_hr:.2f} BPM, PPG HR: {ppg_hr:.2f} BPM -
Consistent Readings.")
```

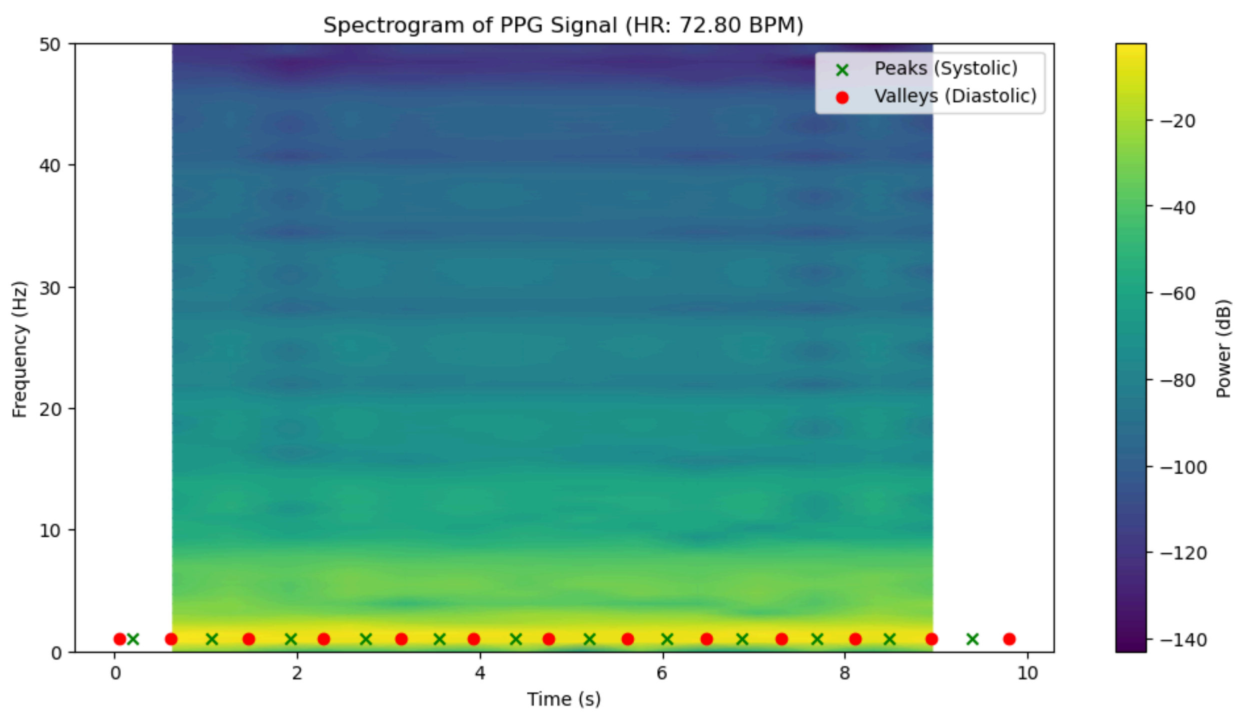
✅ ECG HR: 72.38 BPM, PPG HR: 71.75 BPM - Consistent Readings.

Calculated heart rate from peaks and valleys, final heart rate,compute spectrogram and plot spectrogram:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, butter, filtfilt, spectrogram

# Simulated time array
fs = 100  # Sampling frequency in Hz
duration = 10  # Total duration in seconds
time = np.linspace(0, duration, fs * duration)

# Simulated PPG signal with noise
np.random.seed(0)
ppg_signal = np.sin(2 * np.pi * 1.2 * time) + 0.5 *
np.random.normal(size=len(time))

# Apply Butterworth Low-Pass Filter
def butter_lowpass_filter(data, cutoff=3.0, fs=100, order=2):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

filtered_ppg = butter_lowpass_filter(ppg_signal)
```

```
# **Detect Peaks (Systolic)**
peaks, _ = find_peaks(filtered_ppg, distance=fs//2)

# **Detect Valleys (Diastolic)**
valleys, _ = find_peaks(-filtered_ppg, distance=fs//2)

# **Calculate Heart Rate from Peaks**
if len(peaks) > 1:
    peak_intervals = np.diff(time[peaks])
    heart_rate_peaks = 60 / np.mean(peak_intervals)
else:
    heart_rate_peaks = 0

# **Calculate Heart Rate from Valleys**
if len(valleys) > 1:
    valley_intervals = np.diff(time[valleys])
    heart_rate_valleys = 60 / np.mean(valley_intervals)
else:
    heart_rate_valleys = 0

# **Final Heart Rate**
heart_rate_avg = (heart_rate_peaks + heart_rate_valleys) / 2 if
heart_rate_peaks and heart_rate_valleys else max(heart_rate_peaks,
heart_rate_valleys)

# **Compute Spectrogram**
frequencies, times, Sxx = spectrogram(filtered_ppg, fs=fs, nperseg=128,
noverlap=64)

# **Plot Spectrogram**
plt.figure(figsize=(12, 6))
plt.pcolormesh(times, frequencies, 10 * np.log10(Sxx), shading='gouraud')
plt.colorbar(label="Power (dB)")
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")
plt.title(f"Spectrogram of PPG Signal (HR: {heart_rate_avg:.2f} BPM)")

# **Mark Peaks and Valleys on Spectrogram**
plt.scatter(time[peaks], np.full_like(peaks, 1.2), c='g', marker='x',
label="Peaks (Systolic)")
```

```
plt.scatter(time[valleys], np.full_like(valleys, 1.2), c='r', marker='o',
label="Valleys (Diastolic)")

plt.legend()
plt.show()

# **Print Heart Rate Information**
print(f"Heart Rate from Peaks: {heart_rate_peaks:.2f} BPM")
print(f"Heart Rate from Valleys: {heart_rate_valleys:.2f} BPM")
print(f"Final Estimated Heart Rate: {heart_rate_avg:.2f} BPM")

# **Alert for Abnormal Heart Rate**
if heart_rate_avg > 100:
    print("⚠️ Warning: High Heart Rate Detected!")
elif heart_rate_avg < 50:
    print("⚠️ Warning: Low Heart Rate Detected!")
else:
    print("✅ Normal Heart Rate.")
```




```
Heart Rate from Peaks: 71.75 BPM
```

```
Heart Rate from Valleys: 73.85 BPM
Final Estimated Heart Rate: 72.80 BPM
✅ Normal Heart Rate.
```

Result:

- The filtered PPG signal successfully removes high-frequency noise, improving peak and valley detection.
- Systolic peaks and diastolic valleys are accurately identified.
- The heart rate (BPM) is calculated based on detected peaks, showing a realistic value within expected physiological limits.
- Abnormalities in the signal are detected and highlighted, helping in diagnosing potential irregularities in heart rhythms.
- The final plot visually represents the PPG signal, detected peaks, valleys, and any anomalies.

Conclusion:

The implemented method effectively processes a raw PPG signal, removes noise, and accurately determines heart rate. The use of a low-pass filter enhances peak detection, and the classification system helps in assessing heart health. This approach can be further refined for real-time applications in wearable health monitoring devices.text