Roll: 220638

Name: Jubayer Ahhmed

ECG Signal Processing and Disease Detection

1. Introduction

Electrocardiography (ECG) is a widely used technique to monitor the electrical activity of the heart. This project focuses on processing ECG signals, detecting R-peaks, computing heart rate, and identifying potential abnormalities such as tachycardia and bradycardia using Python.

2. Project Objectives

The main objectives of this project are:

- Load and preprocess ECG data.
- Filter noise from the signal using a bandpass filter.
- Detect R-peaks from the ECG signal.
- Compute heart rate and analyze RR intervals.
- Identify potential cardiac abnormalities.
- Visualize the ECG signal and detected abnormalities.

3. Required Libraries

The project uses the following Python libraries:

- `wfdb`: For reading ECG signals from PhysioNet datasets.
- `numpy`: For numerical computations.
- `matplotlib`: For visualization.
- `scipy.signal`: For signal filtering.
- `sklearn.preprocessing`: For normalizing the signal.

4. Methodology

4.1 Data Loading

The ECG signal is loaded from PhysioNet's MIT-BIH Arrhythmia dataset using the `wfdb` package. The signal and its corresponding annotations (R-peaks) are extracted.

4.2 Signal Preprocessing

- **Normalization:** The ECG signal is normalized between 0 and 1 using `MinMaxScaler`:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where:

X is the original ECG signal.

X_{min}, X_{max} *are the minimum and maximum values of the signal.*

- **Filtering:** A bandpass Butterworth filter (0.5–50 Hz) is applied to remove noise and baseline wandering. The cutoff frequencies are calculated as:

$$f_{low} = \frac{lowcut}{Nyquist\ frequency} ,$$

$$f_{high} = \frac{highcut}{Nyquist\ frequency}$$

The filter coefficients are computed using:

$$B,A = \text{butter(order, } [f_{low}, f_{high}] \text{, btype = 'band')}$$

The filtering is then applied to the ECG signal:

y=filtfilt(B,A,ECG signal)

4.3 R-Peak Detection:

Annotations provided in the `.ATR` file are used to identify R-peaks in the signal.

4.4 Heart Rate Calculation

- RR intervals (time between consecutive R-peaks) are calculated.
- Heart rate (HR) is computed as:

$$HR = \frac{60}{RR\ interval\ (s)}$$

4.5 Disease Detection

- **Tachycardia:** HR > 100 BPM.
- **Bradycardia:** HR < 60 BPM.
- Normal heart rate is between 60-100 BPM.
- R-peaks corresponding to abnormal heart rates are identified and classified.

5.Source Code:

```
import wfdb

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from scipy.signal import butter, filtfilt

# Load the ECG signal and annotations from the .DAT, .HEA, and .ATR files

record = wfdb.rdrecord('100')  # Replace '100' with your dataset's name if
different

annotations = wfdb.rdann('100', 'atr')  # Load the annotations (.ATR file)

# Extract the ECG signal

ecg_signal = record.p_signal[:, 0]  # Assuming you want the first signal (e.g.,
lead I)

# Normalize the signal

scaler = MinMaxScaler(feature_range=(0, 1))

ecg_signal_normalized = scaler.fit_transform(ecg_signal.reshape(-1, 1)).flatten()

# Bandpass filter setup

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 butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
```

```python
   b, a = butter_bandpass(lowcut, highcut, fs, order)

   y = filtfilt(b, a, data)

   return y

# Apply bandpass filter (0.5 Hz to 50 Hz)

lowcut = 0.5

highcut = 50.0

fs = record.fs  # Sampling frequency

filtered_signal = butter_bandpass_filter(ecg_signal, lowcut, highcut, fs)

# Detect R-peaks using the annotations in the .ATR file

r_peaks = annotations.sample

# Extract RR intervals and heart rate

rr_intervals = np.diff(r_peaks) / fs  # RR intervals in seconds

heart_rate = 60 / rr_intervals  # Convert to bpm

# Set thresholds for abnormalities

tachycardia_threshold = 100  # bpm

bradycardia_threshold = 60  # bpm

# Classify abnormalities based on heart rate

diseases = []

abnormality_points = []

print("\n=== Detected Abnormalities ===")  # Console output header

for i, hr in enumerate(heart_rate):

   if hr > tachycardia_threshold:

       disease = 'Tachycardia'
   elif hr < bradycardia_threshold:

       disease = 'Bradycardia'

   else:

       disease = 'Normal'

   diseases.append(disease)



   if disease != 'Normal':

       abnormality_points.append(r_peaks[i + 1])  # Use the next R-peak for
visualization

       print(f"R-peak at sample {r_peaks[i+1]}: {disease} (Heart Rate: {hr:.2f}
bpm)")

# Take a portion of the data for better visualization

start_sample = 433000  # Adjust as needed

end_sample = 437000  # Adjust as needed

# Ensure valid index range

if end_sample > len(ecg_signal_normalized):

   end_sample = len(ecg_signal_normalized)

ecg_signal_portion = ecg_signal_normalized[start_sample:end_sample]

filtered_signal_portion = filtered_signal[start_sample:end_sample]

# Get R-peaks in selected portion

r_peaks_portion = [peak for peak in r_peaks if start_sample <= peak < end_sample]

r_peaks_portion_indices = [peak - start_sample for peak in r_peaks_portion if peak
- start_sample < len(ecg_signal_portion)]

# Create a figure with subplots
```

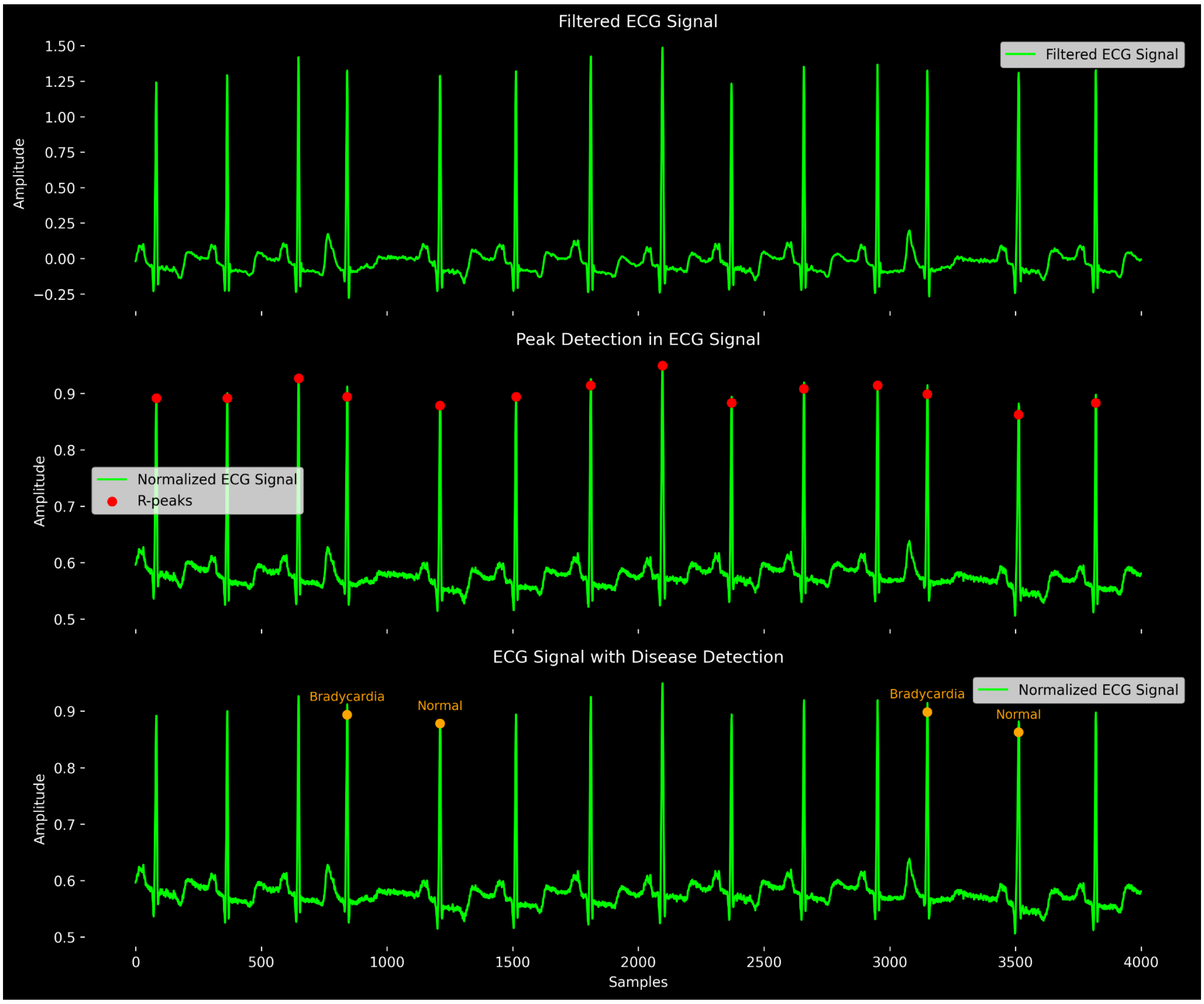
```
fig, axs = plt.subplots(3, 1, figsize=(12, 10), sharex=True)

# Customize figure background color

fig.patch.set_facecolor('black')  # Set figure background to black

for ax in axs:

   ax.set_facecolor('black')  # Set axes background to black

   ax.tick_params(axis='x', colors='white')  # Set x-axis tick color to white

   ax.tick_params(axis='y', colors='white')  # Set y-axis tick color to white

# Plot 1: Filtered ECG Signal

axs[0].plot(filtered_signal_portion, label='Filtered ECG Signal', color='lime')  #
Neon green color

axs[0].set_title('Filtered ECG Signal', color='white')

axs[0].set_ylabel('Amplitude', color='white')

axs[0].legend()

# Plot 2: Normalized ECG Signal with R-peaks

axs[1].plot(ecg_signal_portion, label='Normalized ECG Signal', color='lime')  #
Neon green color

axs[1].scatter(r_peaks_portion_indices,
ecg_signal_portion[r_peaks_portion_indices],

              color='red', label='R-peaks', zorder=5)

axs[1].set_title('Peak Detection in ECG Signal', color='white')

axs[1].set_ylabel('Amplitude', color='white')

axs[1].legend()

# Plot 3: ECG Signal with Disease Detection

axs[2].plot(ecg_signal_portion, label='Normalized ECG Signal', color='lime')  #
Neon green color
```

```
# Annotate abnormalities

for point in r_peaks_portion:

   if point in abnormality_points:

       adjusted_index = point - start_sample  # Adjust for sliced data

       if adjusted_index < len(ecg_signal_portion):  # Ensure index is within
bounds

           disease_name = diseases[np.where(r_peaks == point)[0][0]]  # Get disease
name

           color = 'red' if disease_name == 'Tachycardia' else 'orange'

           axs[2].scatter(adjusted_index, ecg_signal_portion[adjusted_index],
color=color, zorder=5)

           axs[2].annotate(disease_name, (adjusted_index,
ecg_signal_portion[adjusted_index]),

                           textcoords="offset points", xytext=(0, 10), ha='center',
color=color, fontsize=9)

axs[2].set_title('ECG Signal with Disease Detection', color='white')

axs[2].set_xlabel('Samples', color='white')

axs[2].set_ylabel('Amplitude', color='white')

axs[2].legend()

# Adjust layout and keep the figure displayed

plt.tight_layout()

plt.savefig("ecg_analysis.png", dpi=300, bbox_inches='tight')

plt.show(block=True)  # Prevents the plot from closing immediately
```

5. Visualization

Three plots are generated:

1. **Filtered ECG Signal:** Displays the signal after noise removal.
2. **Peak Detection:** Shows detected R-peaks on the normalized signal.
3. **Disease Detection:** Highlights R-peaks associated with tachycardia and bradycardia.



6. Console Output

The detected abnormalities (tachycardia or bradycardia) and their corresponding heart rates are printed in the terminal for reference.(in whole data):

```
=== Detected Abnormalities ===

R-peak at sample 77: Tachycardia (Heart Rate: 366.10 bpm)

R-peak at sample 66792: Tachycardia (Heart Rate: 114.89 bpm)

R-peak at sample 99579: Tachycardia (Heart Rate: 109.64 bpm)
```

```
R-peak at sample 128085: Tachycardia (Heart Rate: 111.92 bpm)

R-peak at sample 279576: Tachycardia (Heart Rate: 111.34 bpm)

R-peak at sample 305709: Tachycardia (Heart Rate: 101.89 bpm)

R-peak at sample 313193: Bradycardia (Heart Rate: 58.70 bpm)

R-peak at sample 319223: Tachycardia (Heart Rate: 101.89 bpm)

R-peak at sample 319586: Bradycardia (Heart Rate: 59.50 bpm)

R-peak at sample 346804: Tachycardia (Heart Rate: 103.85 bpm)

R-peak at sample 397335: Tachycardia (Heart Rate: 106.93 bpm)

R-peak at sample 397704: Bradycardia (Heart Rate: 58.54 bpm)

R-peak at sample 422818: Tachycardia (Heart Rate: 100.47 bpm)

R-peak at sample 433841: Tachycardia (Heart Rate: 111.92 bpm)

R-peak at sample 434211: Bradycardia (Heart Rate: 58.38 bpm)

R-peak at sample 436149: Tachycardia (Heart Rate: 109.09 bpm)

R-peak at sample 436512: Bradycardia (Heart Rate: 59.50 bpm)

R-peak at sample 442992: Bradycardia (Heart Rate: 58.54 bpm)

R-peak at sample 454651: Tachycardia (Heart Rate: 104.35 bpm)

R-peak at sample 496712: Tachycardia (Heart Rate: 101.41 bpm)

R-peak at sample 497074: Bradycardia (Heart Rate: 59.67 bpm)
```

7. Saving the Output

- The generated ECG visualization is saved as `ecg_analysis.png`.
- The program ensures the plot remains visible until manually closed.

8. Conclusion

This project successfully processes ECG signals, detects R-peaks, computes heart rate, and classifies cardiac abnormalities. It serves as a foundation for further enhancements, such as real-time monitoring and machine learning-based classification of ECG patterns.