191805057 – Burak Tüzel PQRST Block Detector

Code and Graph Explanation

İmporting electrocardiogram data from scipy.misc, declaring the frequency as fs, declaring lower and upper bounds, finding the time beggining to end of the signal then using gaussian smoothing function for smoothing the signal

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import electrocardiogram
from scipy.signal import find_peaks

def derivative_filter(signal):
    return np.diff(signal)


ecg_signal = electrocardiogram()
fs = 360  #Frequency

#Lower and Upper bounds
lower = 2150
upper = 3050
#Signal time
time = np.arange(ecg_signal.size) / fs

from scipy.signal import windows

def gaussian_smooth(data, sigma=2):
    window = windows.gaussian(5*sigma+5, sigma)
    smoothed_data = np.convolve(data, window, mode='same') / window.sum()
    return smoothed_data
#Smoothing the signal
smoothed_ecg_signal = gaussian_smooth(ecg_signal)
```

Applying the baseline correction to to separate true spectroscopic signals from interference effects and finding r peaks for smoothed ecg signal, then declaring time_lower and time_upper to use as limits for graphs

```python
def baseline_correction(signal, window_size):
    rolling_mean = np.convolve(signal, np.ones(window_size)/window_size, mode='same')
    corrected_signal = signal - rolling_mean
    return corrected_signal
#Baseline correction
smoothed_ecg_signal = baseline_correction(smoothed_ecg_signal, window_size=100)
#Smoothed signal time
time_secg = np.arange(len(smoothed_ecg_signal)) / fs

#Function to detect R peaks in a signal
def detect_r_peaks(signal, threshold, neighborhood=5):
    peaks, _ = find_peaks(signal, height=threshold)
    filtered_peaks = []
    for peak in peaks:
        if len(filtered_peaks) == 0 or peak - filtered_peaks[-1] > neighborhood:
            filtered_peaks.append(peak)
        elif signal[peak] > signal[filtered_peaks[-1]]:
            filtered_peaks[-1] = peak
    return np.array(filtered_peaks)


threshold_filtered = 0.15 * max(smoothed_ecg_signal)
r_peaks_secg = detect_r_peaks(smoothed_ecg_signal, threshold_filtered, neighborhood=120)

#Lower and Upper Bound in time (limiting with it)
time_lower = lower / fs
time_upper = upper / fs
```

Detecting P,Q,S,T keypoints depending on R Peak with this function

```python
#Detecting other keypoints
def detect_p_q_s_t_points(signal, r_peaks, fs):
    p_points = []
    q_points = []
    s_points = []
    t_points = []

    for r_peak in r_peaks:

        search_window_p_start = max(0, r_peak - int(0.2 * fs))
        search_window_p_end = min(r_peak, r_peak - int(0.07 * fs))

        p_point = np.argmax(signal[search_window_p_start:search_window_p_end]) + search_window_p_start

        p_points.append(p_point)

        search_window_start = max(0, r_peak - int(0.04* fs))
        search_window_end = min(len(signal), r_peak + int(0.04 * fs))

        q_point = np.argmin(signal[search_window_start:r_peak]) + search_window_start
        s_point = np.argmin(signal[r_peak:search_window_end]) + r_peak

        search_window_t_start = s_point
        search_window_t_end = min(len(signal), s_point + int(0.2 * fs))

        # Find T-point by looking for the maximum value in the search window
        t_point = np.argmax(signal[search_window_t_start:search_window_t_end]) + search_window_t_start

        q_points.append(q_point)
        s_points.append(s_point)
        t_points.append(t_point)

    return np.array(p_points), np.array(q_points), np.array(s_points), np.array(t_points)
```

Finding keypoint times between the point lower and upper time bounds with this function

```python
#Finding Keypoint times between lower and upper bound times
def find_keypoint_times(p_point,q_point,r_point,s_point,t_point,fs,lower,upper):
    p_point_times = []
    q_point_times = []
    r_point_times = []
    s_point_times = []
    t_point_times = []

    for point in p_point:
        point_time = time_lower + point/fs
        if((point >= lower) & (point <= upper)):
            p_point_times.append(point_time)


    for point in q_point:
        point_time = time_lower + point/fs
        if((point >= lower) & (point <= upper)):
            q_point_times.append(point_time)

    for point in r_point:
        point_time = time_lower + point/fs
        if((point >= lower) & (point <= upper)):
            r_point_times.append(point_time)

    for point in s_point:
        point_time = time_lower + point/fs
        if((point >= lower) & (point <= upper)):
            s_point_times.append(point_time)

    for point in t_point:
        point_time = time_lower + point/fs
        if((point >= lower) & (point <= upper)):
            t_point_times.append(point_time)
    point_iter = len(p_point_times)
    print(point_iter)
    if((len(p_point_times) == point_iter) & (len(q_point_times) == point_iter) & (len(r_point_times) == point_iter) & (len(s
        return p_point_times,q_point_times,r_point_times,s_point_times,t_point_times,point_iter
#Calculating intervals depending on keypoint times
```

Printing the intervals between the points and point blocks as you can see below with this function

```
#Calculating intervals depending on keypoint times
def calculate_intervals():
    p_point_times,q_point_times,r_point_times,s_point_times,t_point_times,point_iter = find_keypoint_times(p_points, q_points
    for i in range(0, point_iter):

        print("*******************************")
        print("** BLOCK {0} PQ INTERVAL: ".format(i+1) + f"{q_point_times[i]-p_point_times[i]:0.3f} **")
        print("** BLOCK {0} QR INTERVAL: ".format(i+1) + f"{r_point_times[i]-q_point_times[i]:0.3f} **")
        print("** BLOCK {0} RS INTERVAL: ".format(i+1) + f"{s_point_times[i]-r_point_times[i]:0.3f} **")
        print("** BLOCK {0} ST INTERVAL: ".format(i+1) + f"{t_point_times[i]-s_point_times[i]:0.3f} **")
        print("** BLOCK {0} PT INTERVAL: ".format(i+1) + f"{t_point_times[i]-p_point_times[i]:0.3f} **")
        print("*******************************")

    for i in range(0, point_iter-1):
        print("*****************************************")
        print("** BLOCK {0} TO BLOCK {1} PP INTERVAL: ".format(i+1, i+2) + f"{p_point_times[i+1]-p_point_times[i]:0.3f} **")
        print("** BLOCK {0} TO BLOCK {1} QQ INTERVAL: ".format(i+1, i+2) + f"{q_point_times[i+1]-q_point_times[i]:0.3f} **")
        print("** BLOCK {0} TO BLOCK {1} TP INTERVAL: ".format(i+1, i+2) + f"{p_point_times[i+1]-t_point_times[i]:0.3f} **")
        print("*****************************************")
#using detecting keypoints function
```

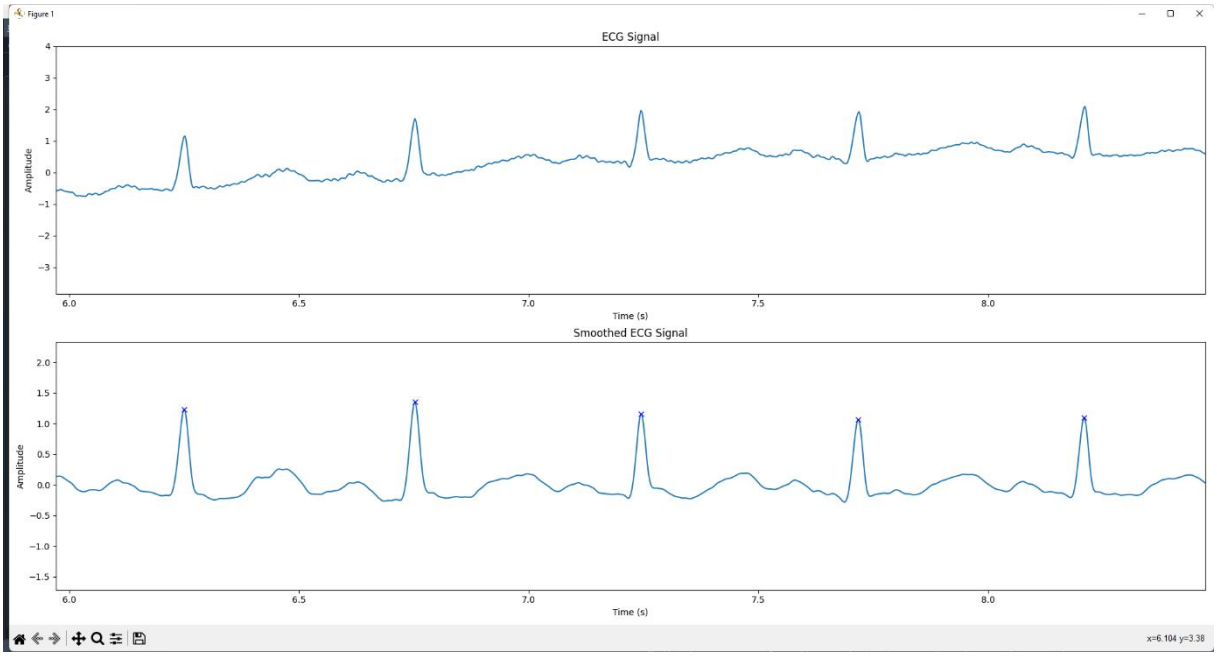
Output

```
*******************************
** BLOCK 1 PQ INTERVAL: 0.111 **
** BLOCK 1 QR INTERVAL: 0.033 **
** BLOCK 1 RS INTERVAL: 0.036 **
** BLOCK 1 ST INTERVAL: 0.172 **
** BLOCK 1 PT INTERVAL: 0.353 **
*******************************
*******************************
** BLOCK 2 PQ INTERVAL: 0.092 **
** BLOCK 2 QR INTERVAL: 0.033 **
** BLOCK 2 RS INTERVAL: 0.033 **
** BLOCK 2 ST INTERVAL: 0.192 **
** BLOCK 2 PT INTERVAL: 0.350 **
*******************************
*******************************
** BLOCK 3 PQ INTERVAL: 0.108 **
** BLOCK 3 QR INTERVAL: 0.028 **
** BLOCK 3 RS INTERVAL: 0.036 **
** BLOCK 3 ST INTERVAL: 0.194 **
** BLOCK 3 PT INTERVAL: 0.367 **
*******************************
*******************************
** BLOCK 4 PQ INTERVAL: 0.106 **
** BLOCK 4 QR INTERVAL: 0.031 **
** BLOCK 4 RS INTERVAL: 0.028 **
** BLOCK 4 ST INTERVAL: 0.197 **
** BLOCK 4 PT INTERVAL: 0.361 **
*******************************
*******************************
** BLOCK 5 PQ INTERVAL: 0.106 **
** BLOCK 5 QR INTERVAL: 0.028 **
** BLOCK 5 RS INTERVAL: 0.036 **
** BLOCK 5 ST INTERVAL: 0.194 **
** BLOCK 5 PT INTERVAL: 0.364 **
*******************************
*****************************************
** BLOCK 1 TO BLOCK 2 PP INTERVAL: 0.522 **
** BLOCK 1 TO BLOCK 2 QQ INTERVAL: 0.503 **
** BLOCK 1 TO BLOCK 2 TP INTERVAL: 0.169 **
*****************************************
*****************************************
** BLOCK 2 TO BLOCK 3 PP INTERVAL: 0.481 **
** BLOCK 2 TO BLOCK 3 QQ INTERVAL: 0.497 **
** BLOCK 2 TO BLOCK 3 TP INTERVAL: 0.131 **
*****************************************
*****************************************
** BLOCK 3 TO BLOCK 4 PP INTERVAL: 0.472 **
** BLOCK 3 TO BLOCK 4 QQ INTERVAL: 0.469 **
** BLOCK 3 TO BLOCK 4 TP INTERVAL: 0.106 **
*****************************************
*****************************************
** BLOCK 4 TO BLOCK 5 PP INTERVAL: 0.494 **
** BLOCK 4 TO BLOCK 5 QQ INTERVAL: 0.494 **
** BLOCK 4 TO BLOCK 5 TP INTERVAL: 0.133 **
*****************************************
```

Plotting the signal in the end of the code

```
#using detecting keypoints function
p_points,q_points,s_points, t_points = detect_p_q_s_t_points(smoothed_ecg_signal , r_peaks_secg, fs)

#using calcule intervals
calculate_intervals()


# Plotting the Raw ECG-Signal
plt.figure(figsize=(20, 10))

# Plot the ECG Signal
plt.subplot(2, 1, 1)
plt.plot(time, ecg_signal)
plt.title('ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.xlim(time_lower, time_upper)

# Ploting the Smoothed ECG-Signal
plt.subplot(2, 1, 2)
plt.plot(time_secg, smoothed_ecg_signal)
plt.plot(r_peaks_secg/fs, smoothed_ecg_signal[r_peaks_secg], 'bx')
plt.title('Smoothed ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.xlim(time_lower, time_upper)

plt.tight_layout()
plt.show()
```

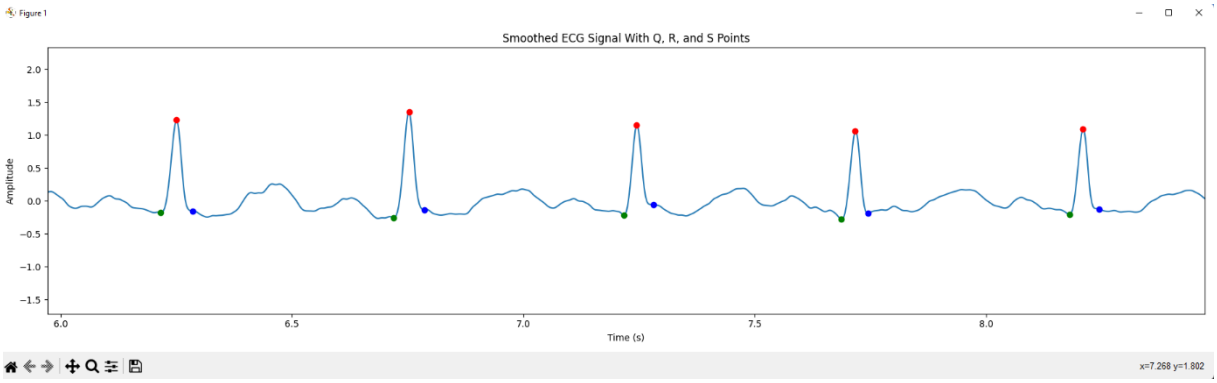
Output: Raw ECG Signal and Smoothed ECG Signal with R Peaks



```python
# Plot the Smoothed ECG Signal with Q,R,S Keypoints
plt.figure(figsize=(20, 5))
plt.plot(time_secg, smoothed_ecg_signal)
plt.plot(r_peaks_secg/fs, smoothed_ecg_signal[r_peaks_secg], 'ro')  # Mark R peaks in red
plt.plot(q_points/fs, smoothed_ecg_signal[q_points], 'go')  # Mark Q points in green
plt.plot(s_points/fs, smoothed_ecg_signal[s_points], 'bo')  # Mark S points in blue
plt.title('Smoothed ECG Signal With Q, R, and S Points')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.xlim(time_lower, time_upper)

plt.tight_layout()
plt.show()
```

Output: Smoothed ECG Signal with Q,R and S points

Q = green dot , R = red dot , S = blue dot



```
# Plot the Smoothed ECG Signal with P,Q,R,S and T points
plt.figure(figsize=(15, 4))
plt.plot(time_secg, smoothed_ecg_signal)
plt.plot(r_peaks_secg/fs, smoothed_ecg_signal[r_peaks_secg], 'ro')  # Mark R peaks in red
plt.plot(q_points/fs, smoothed_ecg_signal[q_points], 'go')  # Mark Q points in green
plt.plot(s_points/fs, smoothed_ecg_signal[s_points], 'bo')  # Mark S points in blue
plt.plot(p_points/fs, smoothed_ecg_signal[p_points], 'mo')  # Mark P points in magenta
plt.plot(t_points/fs, smoothed_ecg_signal[t_points], 'yo')  # Mark P points in magenta
plt.title('Smoothed ECG Signal With P, Q, R, S, T Points')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.xlim(time_lower, time_upper)

plt.tight_layout()
plt.show()
```

Output: Smoothed ECG Signal with P, Q, R, S and T Keypoints

P = magenta dots, Q = green dots, R = red dots, S = blue dots, T = yellow dots