**PYTHON CODE FOR ECG ACQUISTION,PRE-PROCESSING AND HRV ANALYSIS**

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import butter, filtfilt, lfilter, find\_peaks

import serial

import time

# --------------------------- Serial Data Acquisition --------------------------- #

# ✅ Serial Connection to Arduino

SERIAL\_PORT = "COM4"  # Change this

BAUD\_RATE = 9600

DURATION = 300 # seconds

fs = 250  # Hz

try:

    ser = serial.Serial(SERIAL\_PORT, BAUD\_RATE, timeout=1)

    print("✅ Connected to Arduino!")

except Exception as e:

    print(f"❌ Error connecting to Arduino: {e}")

    exit()

# ✅ Data Collection

total\_samples = fs \* DURATION

ecg\_data = []

print("⏳ Collecting ECG data...")

start\_time = time.time()

while len(ecg\_data) < total\_samples:

    try:

        line = ser.readline().decode('utf-8').strip()

        if line.isdigit():

            ecg\_data.append(int(line))

    except Exception as e:

        print(f"⚠️ Serial Read Error: {e}")

    if time.time() - start\_time > DURATION:

        print("⚠️ Timeout reached.")

        break

ser.close()

print("✅ Data collection complete!")

ecg\_data = np.array(ecg\_data)

# --------------------------- Preprocessing --------------------------- #

# ✅ Bandpass Filter Function

def butter\_bandpass(lowcut, highcut, fs, order=4):

    nyquist = 0.5 \* fs

    low = lowcut / nyquist

    high = highcut / nyquist

    b, a = butter(order, [low, high], btype='band')

    return b, a

def apply\_bandpass\_filter(data, lowcut=0.5, highcut=35.0, fs=250, order=4):

    if len(data) == 0:

        raise ValueError("❌ ECG data is empty! Check serial communication.")

    b, a = butter\_bandpass(lowcut, highcut, fs, order=order)

    if len(data) < max(len(a), len(b)) \* 3:

        print("⚠️ Data too short for filtfilt, using lfilter instead.")

        return lfilter(b, a, data)  # Use lfilter for short signals

    return filtfilt(b, a, data)

# ✅ Moving Average Smoothing

def moving\_average(data, window\_size=7):

    return np.convolve(data, np.ones(window\_size) / window\_size, mode='same')

# ✅ Detect R-Peaks on smoothed signal

def detect\_r\_peaks\_direct(ecg\_signal, fs=250, height\_ratio=0.4, distance\_sec=0.3):

    threshold = height\_ratio \* np.max(ecg\_signal)

    min\_distance = int(distance\_sec \* fs)

    peaks, \_ = find\_peaks(ecg\_signal, height=threshold, distance=min\_distance)

    return peaks

# --------------------------- Processing Pipeline --------------------------- #

# ✅ Step 1: Bandpass Filtering

ecg\_filtered = apply\_bandpass\_filter(ecg\_data, lowcut=0.5, highcut=35.0, fs=fs, order=4)

# ✅ Step 2: Moving Average Smoothing

ecg\_smoothed = moving\_average(ecg\_filtered, window\_size=7)

# ✅ Step 3: Centering

ecg\_centered = ecg\_smoothed - np.mean(ecg\_smoothed)

# ✅ Step 4: R-Peak Detection

r\_peaks = detect\_r\_peaks\_direct(ecg\_centered, fs=fs, height\_ratio=0.4, distance\_sec=0.3)  # You can tune these values

# --------------------------- Plotting --------------------------- #

# ✅ Plot raw ECG

plt.figure(figsize=(14, 5))

plt.plot(ecg\_data, color='blue')

plt.title('Raw ECG Signal')

plt.xlabel('Sample Number')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

# ✅ Optional: Show different stages for understanding

plt.figure(figsize=(16, 8))

plt.subplot(3, 1, 1)

plt.plot(ecg\_filtered[:1000], color='blue', label='Bandpass Filtered')

plt.title("Bandpass Filtered ECG (0.5 - 35 Hz)")

plt.ylabel("Amplitude")

plt.legend()

plt.grid(True)

plt.subplot(3, 1, 2)

plt.plot(ecg\_centered[:1000], color='orange', label='Smoothed & Centered')

plt.title("Smoothed and Centered ECG")

plt.ylabel("Amplitude")

plt.legend()

plt.grid(True)

plt.subplot(3, 1, 3)

plt.plot(ecg\_centered[:1000], color='green', label='Final Processed ECG')

plt.scatter(r\_peaks[r\_peaks < 1000], ecg\_centered[r\_peaks[r\_peaks < 1000]], color='red', label='Detected R-peaks', zorder=5)

plt.title("Detected R-peaks")

plt.xlabel("Sample Number")

plt.ylabel("Amplitude")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# --------------------------- RR Interval Calculation & Plotting --------------------------- #

# ✅ Calculate RR intervals in samples

rr\_intervals\_samples = np.diff(r\_peaks)  # Difference between successive R-peaks

# ✅ Convert to milliseconds

rr\_intervals\_ms = (rr\_intervals\_samples / fs) \* 1000  # fs = 250 Hz

print("\n✅ RR Intervals (ms):")

print(rr\_intervals\_ms)

# ✅ Time axis for RR intervals (use time of second R-peak onward)

rr\_times = r\_peaks[1:] / fs  # Convert to seconds for plotting

# ✅ RR Interval Plot

plt.figure(figsize=(12, 6))

plt.plot(rr\_times, rr\_intervals\_ms, marker='o', linestyle='-', color='purple')

plt.title("RR Interval vs Time")

plt.xlabel("Time (s)")

plt.ylabel("RR Interval (ms)")

plt.grid(True)

plt.show()

#--------------------------------TIME-DOMAIN HRV METRICS-----------------------------#

# ✅ Mean RR Interval (ms)

mean\_rr = np.mean(rr\_intervals\_ms)

# ✅ Mean HR (bpm) = 60000 / Mean RR

mean\_hr = 60000 / mean\_rr

# ✅ Min and Max HR (bpm)

min\_hr = 60000 / np.max(rr\_intervals\_ms)

max\_hr = 60000 / np.min(rr\_intervals\_ms)

# ✅ SDNN (Standard deviation of RR intervals)

sdnn = np.std(rr\_intervals\_ms)

# ✅ RMSSD (Root Mean Square of Successive Differences)

rmssd = np.sqrt(np.mean(np.diff(rr\_intervals\_ms)\*\*2))

# ✅ NN50 (number of successive RR intervals differing by more than 50ms)

nn50 = np.sum(np.abs(np.diff(rr\_intervals\_ms)) > 50)

# ✅ pNN50 (percentage of NN50)

pnn50 = (nn50 / len(rr\_intervals\_ms)) \* 100

# ✅ Print Time-domain results

print("\n🔵 Time-Domain HRV Results:")

print(f"Mean RR (ms): {mean\_rr:.2f}")

print(f"Mean HR (bpm): {mean\_hr:.2f}")

print(f"Min HR (bpm): {min\_hr:.2f}")

print(f"Max HR (bpm): {max\_hr:.2f}")

print(f"SDNN (ms): {sdnn:.2f}")

print(f"RMSSD (ms): {rmssd:.2f}")

print(f"NN50 (beats): {nn50}")

print(f"pNN50 (%): {pnn50:.2f}")

#------------------------------------FREQUENCY-DOMAIN HRV VIA FFT(POWER SPECTRAL DENSITY)------------------#

from scipy.signal import welch

# ✅ Interpolate RR intervals to get evenly sampled signal for FFT

from scipy.interpolate import interp1d

# Interpolation to get continuous RR signal

times = np.cumsum(rr\_intervals\_ms) / 1000  # convert to seconds

interp\_func = interp1d(times, rr\_intervals\_ms, kind='cubic')

time\_interp = np.linspace(times[0], times[-1], len(times))

rr\_interp = interp\_func(time\_interp)

# ✅ Compute Power Spectrum using Welch's method

frequencies, psd = welch(rr\_interp, fs=4.0, nperseg=256)  # 4Hz interpolation

# ✅ Frequency bands (Hz)

vlf\_band = (0.003, 0.04)

lf\_band = (0.04, 0.15)

hf\_band = (0.15, 0.40)

# ✅ Band powers

vlf\_power = np.trapz(psd[(frequencies >= vlf\_band[0]) & (frequencies < vlf\_band[1])],

                     frequencies[(frequencies >= vlf\_band[0]) & (frequencies < vlf\_band[1])])

lf\_power = np.trapz(psd[(frequencies >= lf\_band[0]) & (frequencies < lf\_band[1])],

                    frequencies[(frequencies >= lf\_band[0]) & (frequencies < lf\_band[1])])

hf\_power = np.trapz(psd[(frequencies >= hf\_band[0]) & (frequencies < hf\_band[1])],

                    frequencies[(frequencies >= hf\_band[0]) & (frequencies < hf\_band[1])])

# ✅ LF/HF Ratio

lf\_hf\_ratio = lf\_power / hf\_power

# ✅ Print Frequency-domain results

print("\n🟢 Frequency-Domain HRV Results:")

print(f"VLF Power: {vlf\_power:.2f} ms²")

print(f"LF Power: {lf\_power:.2f} ms²")

print(f"HF Power: {hf\_power:.2f} ms²")

print(f"LF/HF Ratio: {lf\_hf\_ratio:.2f}")

#---------------NON-LINEAR HRV------------------#

# ✅ Poincare Plot Metrics

rr\_diff = np.diff(rr\_intervals\_ms)

sd1 = np.sqrt(np.var(rr\_diff) / 2)

sd2 = np.sqrt(2 \* np.var(rr\_intervals\_ms) - (np.var(rr\_diff) / 2))

sd\_ratio = sd2 / sd1

print("\n🟡 Nonlinear HRV Results (Poincare):")

print(f"SD1: {sd1:.2f} ms")

print(f"SD2: {sd2:.2f} ms")

print(f"SD2/SD1 Ratio: {sd\_ratio:.2f}")

#----------------------------------------------plots-------------------------------#

# ✅ RR Interval Histogram

plt.figure(figsize=(8, 5))

plt.hist(rr\_intervals\_ms, bins=30, color='skyblue', edgecolor='black')

plt.title("RR Interval Distribution")

plt.xlabel("RR Interval (ms)")

plt.ylabel("Count")

plt.grid(True)

plt.show()

# ✅ RR Spectrum (FFT-based)

plt.figure(figsize=(8, 5))

plt.plot(frequencies, psd, color='purple')

plt.fill\_between(frequencies, psd, where=((frequencies >= vlf\_band[0]) & (frequencies < vlf\_band[1])), color='red', alpha=0.3, label='VLF')

plt.fill\_between(frequencies, psd, where=((frequencies >= lf\_band[0]) & (frequencies < lf\_band[1])), color='green', alpha=0.3, label='LF')

plt.fill\_between(frequencies, psd, where=((frequencies >= hf\_band[0]) & (frequencies < hf\_band[1])), color='blue', alpha=0.3, label='HF')

plt.title("RR Interval Spectrum (FFT PSD)")

plt.xlabel("Frequency (Hz)")

plt.ylabel("PSD (ms²/Hz)")

plt.legend()

plt.grid(True)

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# ✅ Extract RR interval pairs for Poincare Plot

rr\_n = rr\_intervals\_ms[:-1]   # All RR intervals except the last

rr\_n1 = rr\_intervals\_ms[1:]   # All RR intervals except the first

# ✅ Calculate SD1 & SD2

sd1 = np.sqrt(0.5 \* np.var(rr\_n1 - rr\_n))

sd2 = np.sqrt(2 \* np.var(rr\_n1 + rr\_n))

# ✅ Scatter plot for Poincare

plt.figure(figsize=(8, 8))

plt.scatter(rr\_n, rr\_n1, alpha=0.6, color='blue', label='RR Pairs')

# ✅ Mean of RR for centering ellipse

mean\_rr\_n = np.mean(rr\_n)

mean\_rr\_n1 = np.mean(rr\_n1)

# ✅ Plot identity line

plt.plot([min(rr\_n), max(rr\_n)], [min(rr\_n), max(rr\_n)], 'r--', label='Identity Line')

# ✅ Add SD1 and SD2 lines

plt.plot([mean\_rr\_n - sd1, mean\_rr\_n + sd1], [mean\_rr\_n1 + sd1, mean\_rr\_n1 - sd1], 'g-', linewidth=2, label="SD1")

plt.plot([mean\_rr\_n - sd2, mean\_rr\_n + sd2], [mean\_rr\_n1 - sd2, mean\_rr\_n1 + sd2], 'r-', linewidth=2, label="SD2")

# ✅ Draw the ellipse representing SD1 and SD2

ellipse = patches.Ellipse((mean\_rr\_n, mean\_rr\_n1), width=2\*sd2, height=2\*sd1, angle=45,

                          edgecolor='black', facecolor='none', linestyle='-', linewidth=2)

plt.gca().add\_patch(ellipse)

# ✅ Labels and legend

plt.xlabel("RR(n) (ms)")

plt.ylabel("RR(n+1) (ms)")

plt.title("Poincaré Plot of RR Intervals")

plt.legend()

plt.grid()

plt.show()

# ✅ Print SD1 and SD2 values

print(f"SD1: {sd1:.2f} ms")

print(f"SD2: {sd2:.2f} ms")

print(f"SD2/SD1 Ratio: {sd2/sd1:.3f}")