**一、PPG+IMU处理程序**

**顺序**：用 MATLAB 代码先做基础预处理与 IMU特征（可选）；随后把原始或高通后的 PPG、带通后的 IMU 送入 Python 脚本做“心搏模板相减 + ANC”。

**参数**：

（1）心搏带通：0.8–5 Hz；高通：0.1 Hz；IMU 带通：0.5–15 Hz；ANC：mu=5e-4, order=12（可在 8–16 间调整）。

（2）工频：mains=50 或 60。

**对比图**：调用 run\_pipeline(..., make\_demo\_plot=True) 直接弹出 4 行对比子图；或单独调用 plot\_comparison(...)。

**落地**：若你的数据存在 csv/mat 文件中，读取后以 ppg (N,), acc (N,3), gyro (N,3) 传入即可。若需要，我可以按你的字段名改好读取脚本。

1. **Matlab: PPG&IMU 预处理(采样率100Hz)(** **ppg\_imu\_preprocess\_100Hz.m)**

%% PPG & (3-axis) IMU preprocessing @100 Hz (accelerometer-only compatible)

% 功能：

% - PPG：工频陷波 -> 高通去漂移 -> （用于心率估计的）心搏带通

% - IMU：支持仅三轴 IMU（仅加速度）。去重力/低频 -> 0.5–15 Hz 带通 -> jerk/能量

% 使用：

% % 若只有三轴加速度：把 gyro 传 []

% [ppg\_hp, ppg\_bpHR, imuOut] = ppg\_imu\_preprocess\_100Hz(ppg, acc, [], 100, 50);

% % 若有陀螺仪：

% [ppg\_hp, ppg\_bpHR, imuOut] = ppg\_imu\_preprocess\_100Hz(ppg, acc, gyro, 100, 50);

% 输入：

% ppg : Nx1 原始 PPG 序列

% acc : Nx3 加速度 (x,y,z)

% gyro : Nx3 陀螺仪 (x,y,z)；若没有，传 []

% fs : 采样率（建议 100）

% mains: 工频（50 或 60）

% 输出：

% ppg\_hp : 高通+陷波后的 PPG（去漂移、去工频）

% ppg\_bpHR : 0.8–5 Hz 带通（用于心搏检测/HR 估计）

% imuOut : 结构体，包含 acc\_bp, （可选）gyro\_bp, acc\_mag, （可选）gyro\_mag, jerk\_acc, stEnergy 等

%

% 说明：心搏模板相减与 ANC 放在 Python 代码实现。

function [ppg\_hp, ppg\_bpHR, imuOut] = ppg\_imu\_preprocess\_100Hz(ppg, acc, gyro, fs, mains)

if nargin < 4 || isempty(fs), fs = 100; end

if nargin < 5 || isempty(mains), mains = 50; end

if nargin < 3, gyro = []; end

ppg = ppg(:);

if size(acc,2)~=3, error('acc size must be Nx3'); end

hasGyro = ~isempty(gyro) && size(gyro,2)==3;

%% 1) PPG: 工频陷波 + 高通去漂移 (0.1 Hz)

ppg\_notch = notch\_filter(ppg, fs, mains);

[bh, ah] = butter(2, 0.1/(fs/2), 'high');

ppg\_hp = filtfilt(bh, ah, ppg\_notch);

%% 2) PPG: 心搏带通 (0.8–5 Hz) 仅用于 HR 提取/峰检

[bb, ab] = butter(3, [0.8 5]/(fs/2), 'bandpass');

ppg\_bpHR = filtfilt(bb, ab, ppg\_hp);

%% 3) IMU: 去重力/低频 (高通 ~0.3 Hz) + 带通 0.5–15 Hz

[bhp, ahp] = butter(2, 0.3/(fs/2), 'high');

[bb1, ab1] = butter(3, [0.5 15]/(fs/2), 'bandpass');

acc\_hp = filtfilt(bhp, ahp, acc);

acc\_bp = filtfilt(bb1, ab1, acc\_hp);

if hasGyro

gyro\_hp = filtfilt(bhp, ahp, gyro);

gyro\_bp = filtfilt(bb1, ab1, gyro\_hp);

else

gyro\_bp = [];

end

%% 4) IMU 特征：模长、jerk（仅加速度必有）、短时能量

acc\_mag = sqrt(sum(acc\_bp.^2, 2));

if hasGyro

gyro\_mag = sqrt(sum(gyro\_bp.^2, 2));

else

gyro\_mag = [];

end

% jerk\_acc = 加速度一阶差分（补齐长度）

jerk\_acc = [zeros(1,3); diff(acc\_bp)];

% 短时能量（窗口约 0.5 s）

wlen = round(0.5\*fs); if mod(wlen,2)==0, wlen=wlen+1; end

stEnergy.acc = moving\_energy(acc\_mag, wlen);

if hasGyro

stEnergy.gyro = moving\_energy(gyro\_mag, wlen);

else

stEnergy.gyro = [];

end

%% 输出结构

imuOut.acc\_bp = acc\_bp;

imuOut.acc\_mag = acc\_mag;

imuOut.jerk\_acc = jerk\_acc;

imuOut.stEnergy = stEnergy;

if hasGyro

imuOut.gyro\_bp = gyro\_bp;

imuOut.gyro\_mag = gyro\_mag;

end

end

%% ----------------- 辅助函数 -----------------

function y = moving\_energy(x, w)

% 简单短时能量（平方后滑窗平均）

x2 = x.^2;

win = ones(w,1)/w;

y = filtfilt(win, 1, x2);

end

function y = notch\_filter(x, fs, mains)

% IIR 陷波，Q 约 30，抑制 mains 及其谐波（到 Nyquist）

if nargin<3, mains=50; end

end

1. **Python: 心搏模板相减 + ANC(LMS) + 对比图(anc\_template\_pipeline\_100Hz.py)**

"""

心搏模板相减 + ANC（LMS）流水线 @100 Hz

依赖：numpy, scipy, matplotlib

支持仅三轴 IMU（仅加速度，gyro 可为 None）

使用：

from anc\_template\_pipeline\_100Hz import run\_pipeline

out = run\_pipeline(ppg, acc, gyro=None, fs=100, mains=50, make\_demo\_plot=True)

输入：

ppg: (N,) 原始 PPG

acc: (N,3) 加速度 x,y,z

gyro: (N,3) 陀螺仪 x,y,z 或 None

输出（dict）：

'ppg\_hp' : 高通+陷波后的 PPG

'ppg\_bpHR' : 心搏带通（用于峰检）

'peaks' : 心搏峰索引

'ppg\_template' : 平均心搏模板（标准化长度）

'ppg\_recon' : 基于模板的重构心搏信号

'ppg\_residual' : 模板相减残差

'ppg\_anc' : ANC 后 PPG（对 residual 进一步去运动伪影；若无参考通道，则等于 residual）

'imu\_energy' : {'acc':..., 'gyro':(可选)}

"""

import numpy as np

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

import matplotlib.pyplot as plt

# ----------------- 基础滤波 -----------------

def notch\_filter(x, fs, mains=50, Q=30):

y = np.asarray(x).copy()

max\_harm = int((fs/2)//mains)

for k in range(1, max\_harm+1):

f0 = k\*mains

bw = f0/Q

b, a = iirnotch(w0=f0/(fs/2), Q=f0/bw)

y = filtfilt(b, a, y)

return y

def butter\_highpass(x, fs, fc=0.1, order=2):

b, a = butter(order, fc/(fs/2), btype='high')

return filtfilt(b, a, x)

def butter\_bandpass(x, fs, f1, f2, order=3):

b, a = butter(order, [f1/(fs/2), f2/(fs/2)], btype='band')

# 明确沿时间轴 (axis=0) 滤波，兼容形状 (N,) 或 (N,C)

return filtfilt(b, a, x, axis=0)

# ----------------- 心搏模板相减 -----------------

def heartbeat\_template\_subtraction(ppg, fs=100):

"""返回模板、重构与残差。ppg 需已做去漂移/陷波。

步骤：

1) 对 ppg 做 0.8–5 Hz 带通以检测峰

2) 根据峰分割心搏段，重采样为统一长度，求平均模板

3) 依据相邻 RR 时长对模板缩放/重采样并拼接重构

4) residual = ppg - recon

"""

# 1) 峰检

ppg\_bp = butter\_bandpass(ppg, fs, 0.8, 5.0, order=3)

min\_dist = int(0.3\*fs) # 至少 300 ms 间隔

peaks, \_ = find\_peaks(ppg\_bp, distance=min\_dist, prominence=np.std(ppg\_bp)\*0.5)

if len(peaks) < 4:

# 峰太少，返回零模板

return {

'ppg\_bpHR': ppg\_bp,

'peaks': peaks,

'template': np.zeros(200),

'recon': np.zeros\_like(ppg),

'residual': ppg.copy()

}

# 2) 片段重采样到固定长度（例如 200）

Ntpl = 200

segs = []

for i in range(len(peaks)-1):

a, b = peaks[i], peaks[i+1]

seg = ppg[a:b]

if len(seg) < int(0.25\*fs):

continue

xi = np.linspace(0, 1, len(seg))

xo = np.linspace(0, 1, Ntpl)

seg\_rs = np.interp(xo, xi, seg)

segs.append(seg\_rs)

if len(segs) < 3:

return {

'ppg\_bpHR': ppg\_bp,

'peaks': peaks,

'template': np.zeros(Ntpl),

'recon': np.zeros\_like(ppg),

'residual': ppg.copy()

}

template = np.median(np.vstack(segs), axis=0)

# 3) 依据每个 RR 长度重建

recon = np.zeros\_like(ppg)

for i in range(len(peaks)-1):

a, b = peaks[i], peaks[i+1]

RR = b - a

xo = np.linspace(0, 1, Ntpl)

xi = np.linspace(0, 1, RR)

tpl\_scaled = np.interp(xi, xo, template)

recon[a:b] = tpl\_scaled

residual = ppg - recon

return {

'ppg\_bpHR': ppg\_bp,

'peaks': peaks,

'template': template,

'recon': recon,

'residual': residual

}

# ----------------- 多通道 ANC（LMS / NLMS） -----------------

def nlms\_anc(d, refs, mu=5e-4, order=12, eps=1e-6):

"""归一化 LMS ANC：

d : 目标通道（长度 N）

refs : 参考矩阵 (N, K) ，建议包含 acc 及其一阶差分（可选包含 gyro）

mu : 步长

order: 每个参考通道的 FIR 阶数（总维度 = K\*order）

返回：y\_hat（对 d 的估计噪声），e（残差 = d - y\_hat），W（权）

"""

N, K = refs.shape

if K == 0:

return np.zeros(N), d.copy(), np.zeros((0,))

# 构造延时线：U[t] 形状 (K\*order,)

U = np.zeros((N, K\*order))

for k in range(K):

x = refs[:, k]

for o in range(order):

U[o:N, k\*order+o] = x[:N-o]

w = np.zeros(K\*order)

y\_hat = np.zeros(N)

e = np.zeros(N)

for n in range(N):

u = U[n]

y\_hat[n] = np.dot(w, u)

e[n] = d[n] - y\_hat[n]

norm = np.dot(u, u) + eps

w = w + mu \* e[n] \* u / norm

return y\_hat, e, w

# ----------------- IMU 特征（短时能量） -----------------

return np.convolve(x2, win, mode='same')

# ----------------- 主流程 -----------------

def run\_pipeline(ppg, acc, gyro=None, fs=100, mains=50, make\_demo\_plot=False):

ppg = np.asarray(ppg).astype(float).flatten()

acc = np.asarray(acc, dtype=float).reshape(-1, 3)

gyro\_arr = None if gyro is None or len(np.asarray(gyro).shape)==0 else np.asarray(gyro, dtype=float).reshape(-1, 3)

# 1) PPG 预处理：陷波 + 高通

ppg\_nf = notch\_filter(ppg, fs, mains=mains, Q=30)

ppg\_hp = butter\_highpass(ppg\_nf, fs, fc=0.1, order=2)

# 2) 心搏模板相减

tpl\_out = heartbeat\_template\_subtraction(ppg\_hp, fs=fs)

ppg\_res = tpl\_out['residual']

# 3) ANC 参考：acc 及其一阶差分（可选 gyro）

acc\_bp = butter\_bandpass(acc, fs, 0.5, 15.0, order=3)

accd = np.vstack([np.r\_[0, np.diff(acc\_bp[:, i])] for i in range(3)]).T

refs\_list = [acc\_bp, accd]

gyro\_bp = None

if gyro\_arr is not None:

gyro\_bp = butter\_bandpass(gyro\_arr, fs, 0.5, 15.0, order=3)

gyrod = np.vstack([np.r\_[0, np.diff(gyro\_bp[:, i])] for i in range(3)]).T

refs\_list += [gyro\_bp, gyrod]

refs = np.hstack(refs\_list) if len(refs\_list) > 0 else np.zeros((len(ppg\_res), 0))

# 4) ANC（对 residual 再去运动伪影）

y\_hat, e, \_ = nlms\_anc(ppg\_res, refs, mu=5e-4, order=12, eps=1e-6)

ppg\_anc = e # ANC 输出的残差；若 K=0 则等于 ppg\_res

# 5) IMU 能量（用于可视化/后续门限）

acc\_mag = np.linalg.norm(acc\_bp, axis=1)

en\_acc = short\_time\_energy(acc\_mag, fs)

en\_gyro = None

if gyro\_bp is not None:

gyro\_mag = np.linalg.norm(gyro\_bp, axis=1)

en\_gyro = short\_time\_energy(gyro\_mag, fs)

out = {

'ppg\_hp': ppg\_hp,

'ppg\_bpHR': tpl\_out['ppg\_bpHR'],

'peaks': tpl\_out['peaks'],

'ppg\_template': tpl\_out['template'],

'ppg\_recon': tpl\_out['recon'],

'ppg\_residual': ppg\_res,

'ppg\_anc': ppg\_anc,

'imu\_energy': {'acc': en\_acc, 'gyro': en\_gyro},

'acc\_bp': acc\_bp,

'gyro\_bp': gyro\_bp,

}

if make\_demo\_plot:

plot\_comparison(ppg, out, fs)

return out

# ----------------- 对比图 -----------------

def plot\_comparison(ppg\_raw, out, fs=100):

"""绘制对比图：原始PPG、高通后、模板重构与残差、ANC输出、IMU能量

若无 gyro，则仅绘制 acc 能量。

"""

t = np.arange(len(ppg\_raw))/fs

fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

axes[0].plot(t, ppg\_raw, label='Raw PPG')

axes[0].plot(t, out['ppg\_hp'], alpha=0.7, label='PPG high-pass + notch')

axes[0].set\_title('PPG (raw vs. high-pass)')

axes[0].legend(loc='upper right')

axes[1].plot(t, out['ppg\_hp'], label='PPG HP')

axes[1].plot(t, out['ppg\_recon'], label='Heartbeat template recon')

axes[1].plot(t, out['ppg\_residual'], label='Residual (after template subtraction)')

axes[1].set\_title('Template subtraction')

axes[1].legend(loc='upper right')

axes[2].plot(t, out['ppg\_residual'], label='Residual before ANC')

axes[2].plot(t, out['ppg\_anc'], label='After ANC (final)')

axes[2].set\_title('ANC on residual')

axes[2].legend(loc='upper right')

axes[3].plot(t, out['imu\_energy']['acc'], label='IMU acc energy')

if out['imu\_energy']['gyro'] is not None:

axes[3].plot(t, out['imu\_energy']['gyro'], label='IMU gyro energy')

axes[3].set\_title('IMU short-time energy')

axes[3].set\_xlabel('Time (s)')

axes[3].legend(loc='upper right')

fig.tight\_layout()

plt.show()

# ----------------- 可选：简单合成数据自测（仅三轴加速度） -----------------

if \_\_name\_\_ == '\_\_main\_\_':

fs = 100

N = fs\*30 # 30 s

t = np.arange(N)/fs

# 合成 PPG：1.2 Hz 心率 + 低频漂移 + 噪声 + 两次“吞咽”突变

hr = 1.2

ppg = 0.8\*np.sin(2\*np.pi\*hr\*t) + 0.05\*np.sin(2\*np.pi\*0.05\*t) + 0.02\*np.random.randn(N)

swallow\_idx = [int(8\*fs), int(20\*fs)]

for si in swallow\_idx:

ppg[si:si+int(0.6\*fs)] += np.hstack([np.linspace(0,0.6,int(0.2\*fs)), np.linspace(0.6,0,int(0.4\*fs))])

# 合成三轴加速度：与吞咽同步的短时能量 + 背景小幅随机

acc = 0.02\*np.random.randn(N,3)

for si in swallow\_idx:

acc[si:si+int(0.4\*fs), :] += 0.6\*np.hanning(int(0.4\*fs))[:,None]

out = run\_pipeline(ppg, acc, gyro=None, fs=fs, mains=50, make\_demo\_plot=True)