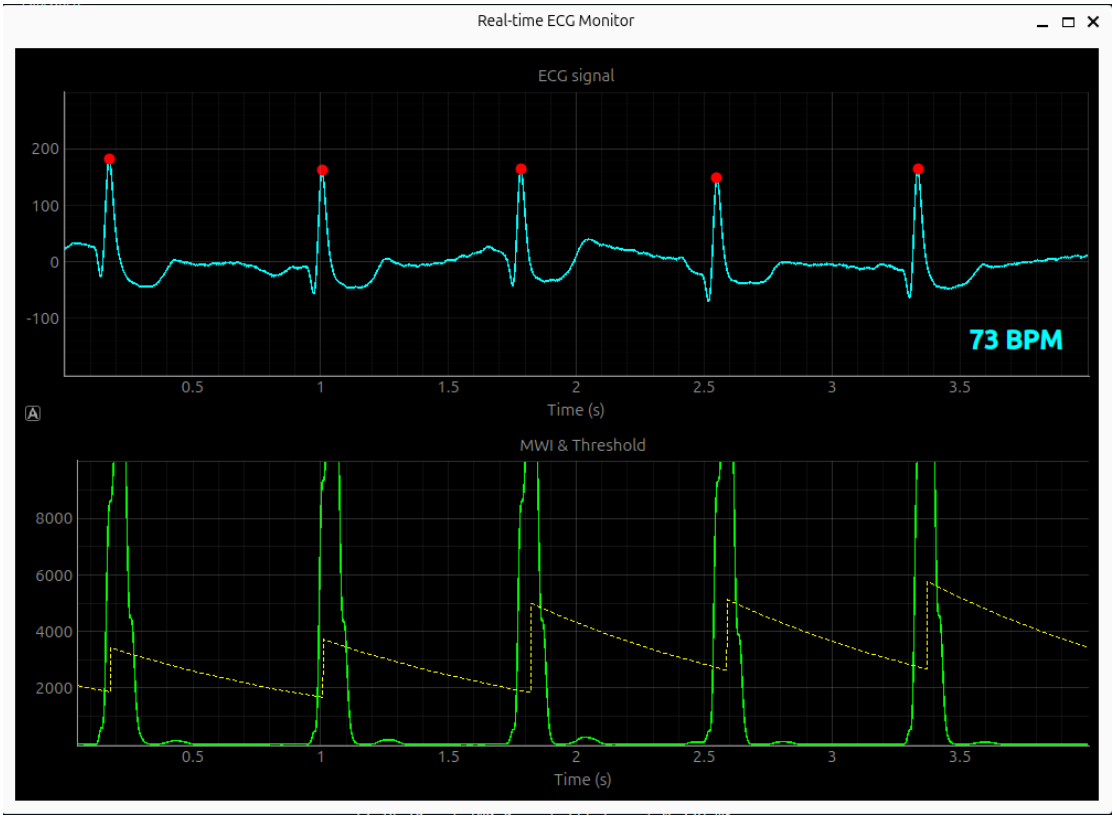
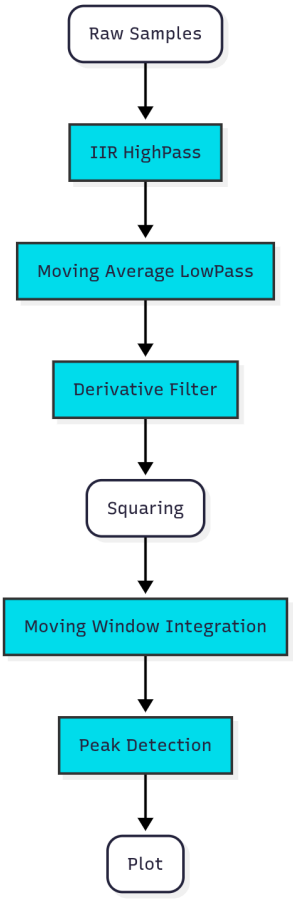
Lab 12: Heart Rate Estimation

111033238 邱泓斌

Peak Detection + Heart Rate Plot



Signal Processing Workflow



IIR HighPass

先將 raw signal 送進 2Hz high pass 濾除低頻的噪聲，如呼吸，身體輕微移動等等。

1-st Order HighPass Filter, 輸出 $y(t)$, 輸入 $x(t)$：

$$\tau\frac{dy(t)}{dt} + y(t) = \tau\frac{dx(t)}{dt}$$

微分用一階差分 discrete：

$$\tau\left(\frac{y[n]-y[n-1]}{T}\right) + y[n] = \tau\left(\frac{x[n]-x[n-1]}{T}\right)$$

$$(y[n]-y[n-1]) + \frac{T}{\tau}y[n] = (x[n]-x[n-1])$$

$$y[n]\left(1+\frac{T}{\tau}\right) = y[n-1] + x[n] - x[n-1]$$

$$y[n] = \frac{1}{\left(1+\frac{T}{\tau}\right)}y[n-1] + \frac{1}{\left(1+\frac{T}{\tau}\right)}(x[n]-x[n-1])$$

定義差分方程：

$$y[n] = \alpha\cdot(y[n-1]+x[n]-x[n-1]),\ \ \alpha = \frac{1}{\left(1+\frac{T}{\tau}\right)}$$

Code 實現：

```
# Highpass (Baseline Wander) config
    dt = 1.0 / fs
    hp_tau = 1.0 / (2 * math.pi * hp_fc)
    hp_alpha = hp_tau / (hp_tau + dt)
---
def _highpass(x, state, a):
    """
    y[n] = a * (y[n-1] + x[n] - x[n-1])
    """
    y_prev = state[0]
    x_prev = state[1]
    y = a * (y_prev + x - x_prev)
    state[0] = y
    state[1] = x
    return y
```

Moving Average Lowpass

為了濾除 60Hz powerline 雜訊，設計了一個 FIR filter - Moving Average Lowpass 來濾除特定頻率的雜訊，推導如下：

Time Domain 上 一個長度為 L 的 Moving Average Filter，其輸出 $y[n]$ 是輸入 $x[n]$ 過去 L 個點的平均值：

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k]$$

Transfer Function:

$$Y(z) = \frac{1}{L}\sum_{k=0}^{L-1} z^{-k}X(z)$$

Then, H(z),

$$H(z) = \frac{1}{L}(1 + z^{-1} + z^{-2} + \cdots + z^{-(L-1)}) = \frac{1}{L}\frac{1-z^{-L}}{1-z^{-1}}$$

Check Frequency Response,

$$H(e^{j\omega}) = \frac{1}{L}\frac{1-e^{-j\omega L}}{1-e^{-j\omega}}$$

Find zeros,

$$1 - e^{-j\omega L} = 0$$
$$\omega L = 2\pi k \ , \ \omega = 2\pi\frac{f}{f_s}$$
$$f = k \cdot \frac{f_s}{L}$$

這次使用 500Hz sample rate 並且要濾除 60Hz 的 powerline noise, 所以設計第一個 zero 要落在差不多的位置，求 window size L:

$$L = \frac{f_s}{60} = \frac{500}{60} \approx 8.33$$

取整的話就是 8 的 window size. 所以我設定 window size = 8 (`ma_len`)

在 moving average filter 做的優化是把原本的 Time Domain 公式改成遞迴的：

$$y[n] = y[n-1] + \frac{1}{L}(x[n] - x[n-L])$$

Code 實現：

```
ma_len: int = 8       # Moving Average Lowpass filter
---
def _mvavg(x, buf, state):
    """
```

```
    S[n] = S[n-1] + x[n] - x[n-L], y[n] = S[n]/L
    y[n] = y[n-1] + (x[n] - x[n-L]) / L
    """
    idx, s = int(state[0]), state[1]
    L = len(buf)
    s += x - buf[idx] # x[n] - x[n-L]
    y = s / L
    buf[idx] = x
    idx = (idx + 1) % L
    state[0], state[1] = idx, s
    return y
```

Derivative Filter

$$d[n] = 2x[n] + x[n-1] + 0 \cdot x[n-2] - x[n-3] - 2x[n-4]$$

透過 derivative filter 進行 highpass 並放大 R peak 的斜率，讓波形更明顯

這裡我設定 deriv_len = 8，理論上 5 就夠了，但保險起見

Code 實現：

```
deriv_len: int = 8    # Derivative filter
---
def _deriv(x, buf, state, sample_idx):
    """
    d[n] = 2x[n] + x[n-1] + 0 x[n-2] - x[n-3] - 2x[n-4]
    """
    if sample_idx < 5:
        return 0.0
    idx = int(state[0])
    n = len(buf)
    buf[idx] = x

    x0 = buf[idx]                # x[n]
    x1 = buf[(idx - 1 + n) % n]    # x[n-1]
    x3 = buf[(idx - 3 + n) % n]    # x[n-3]
    x4 = buf[(idx - 4 + n) % n]    # x[n-4]

    d = (2 * x0 + x1 - x3 - 2 * x4)

    state[0] = (idx + 1) % n
    return d
```

Moving Window Intergration

一樣使用 Moving Average 實現，只是在 Squaring 之後是計算 QRS 波形的能量。給之後 Threshold 判斷超過某個能量值就進 peak detection

因為 sample rate = 500Hz, QRS 波形的 interval 約在 75ms ~ 105ms 之間，因此 mwi 的長度約是 45 = 90ms 可以剛好積分單位 QRS 波區間的能量

Code 實現：

```
mwi_len: int = 45     # Moving window integration
---
def _mwi(x, buf, state):
    """
    S[n] = S[n-1] + x[n] - x[n-L]
    """
    idx, s = int(state[0]), state[1]
    L = len(buf)
    s += x - buf[idx]
    buf[idx] = x
    s = max(0.0, s)
    y = int(s / L)
    idx = (idx + 1) % L
    state[0], state[1] = idx, s
    return y
```

Peak Detection

用了四種機制：

- MWI 積分出的能量要大於 threshold
- 用 slope calculation 算前一刻與下一刻的 slope 是否由正轉負
- Peak 與 Peak 之間的 interval 要大於 200ms
- Threshold 會隨 peak 強度調整，並遞減

Code 實現：

```
def _peak(mwi, state, sample_idx, fs, interval, decay, min_th):
    prev_mwi = state[0]
    prev_slope = state[1]
    th = int(state[2])
    last = int(state[3])

    # Calculate current slope
    slope = mwi - prev_mwi

    # Peak Detect
    samples = int(interval * fs / 1000.0)
    peak = 0.0
```

```
    if (sample_idx - last) > samples:
        if prev_slope > 0 and slope <= 0 and prev_mwi > th:
            peak = 1.0
            last = sample_idx - 1
            th = int(prev_mwi * 0.4)

    # Threshold decay
    th = int(th * decay)
    th = max(th, min_th) # Minimum Threshold

    # Update state
    state[0] = mwi
    state[1] = slope
    state[2] = th
    state[3] = last
    return peak
```

其他優化

除了確保 signal processing 的 funciton 複雜度都是 $O(1)$ 之外，我另外將 Serial Read, Signal Processing, Plot 切成三個 Thread 執行，彼此透過 `queue.Queue` 進行非同步通訊，確保任何單一process的阻塞都不會影響其他部分的工作

最後是 運算效率的呈現

平均處理時間

$$\mathbf{0.4926ms/100}\ \text{samples}$$

所以平均 1 個 samples 大概是 $4.93\ us$ 左右

平均 CPU 佔用

$$\mathbf{87.8\%}$$

單核 100% 佔用 87.8% (含 plot)

平均 RAM 佔用

$$\mathbf{270.2}\ \text{MB}$$

下面是 log 紀錄：

```
[Processor] Compiling Numba functions...
[Processor] JIT Compiled.
[Monitor] Dashboard active (PID: 340904)
Time      | CPU %   | RAM MB   | Proc Time (100pts)
----------------------------------------------------
01:41:06  | 0.0     | 248.1    | Waiting...
01:41:07  | 25.9    | 269.9    | Waiting...
```

```
[Serial] Listening on /dev/ttyACM0...
01:41:08 | 2.0   | 269.9 | Waiting...
01:41:09 | 44.0  | 270.1 | 0.419 ms
01:41:10 | 68.6  | 267.4 | 0.419 ms
01:41:11 | 88.9  | 267.4 | 0.342 ms
01:41:12 | 111.5 | 267.7 | 0.379 ms
01:41:13 | 73.2  | 267.7 | 0.386 ms
01:41:14 | 65.9  | 273.3 | 0.468 ms
01:41:15 | 70.0  | 267.7 | 0.440 ms
01:41:16 | 67.9  | 267.7 | 0.470 ms
01:41:17 | 69.9  | 267.7 | 0.372 ms
01:41:18 | 78.8  | 267.7 | 0.405 ms
01:41:19 | 70.9  | 267.7 | 0.379 ms
01:41:20 | 68.9  | 267.9 | 0.648 ms
01:41:21 | 72.4  | 267.9 | 0.373 ms
01:41:22 | 70.9  | 267.9 | 0.353 ms
01:41:23 | 70.9  | 267.9 | 0.431 ms
01:41:24 | 72.6  | 273.5 | 0.422 ms
01:41:25 | 70.9  | 268.0 | 0.396 ms
01:41:26 | 72.6  | 268.0 | 0.471 ms
01:41:27 | 70.3  | 268.0 | 0.415 ms
01:41:28 | 84.9  | 268.1 | 0.309 ms
01:41:29 | 120.6 | 270.9 | 0.284 ms
01:41:30 | 120.0 | 268.2 | 0.305 ms
01:41:31 | 118.6 | 268.2 | 0.313 ms
01:41:32 | 116.9 | 270.9 | 0.300 ms
01:41:33 | 97.8  | 269.1 | 0.441 ms
01:41:34 | 111.8 | 270.9 | 0.471 ms
01:41:35 | 108.9 | 268.2 | 0.376 ms
01:41:36 | 98.8  | 268.2 | 0.346 ms
01:41:37 | 98.0  | 268.2 | 0.291 ms
01:41:38 | 153.8 | 273.8 | 0.754 ms
01:41:39 | 147.6 | 273.8 | 0.699 ms
01:41:40 | 143.1 | 271.2 | 0.564 ms
01:41:41 | 135.1 | 271.2 | 0.545 ms
01:41:42 | 139.2 | 271.2 | 0.646 ms
01:41:43 | 132.7 | 271.1 | 0.577 ms
01:41:44 | 138.0 | 271.1 | 0.560 ms
01:41:45 | 138.8 | 268.5 | 0.867 ms
01:41:46 | 130.6 | 271.3 | 0.560 ms
01:41:47 | 131.1 | 271.3 | 0.712 ms
01:41:48 | 136.0 | 271.3 | 0.631 ms
01:41:49 | 136.9 | 271.3 | 0.686 ms
01:41:50 | 165.4 | 268.5 | 0.674 ms
01:41:51 | 137.1 | 268.5 | 0.583 ms
01:41:52 | 138.3 | 271.1 | 0.577 ms
01:41:53 | 136.3 | 273.9 | 0.691 ms
01:41:54 | 138.1 | 273.9 | 0.662 ms
```

```
01:41:55  | 127.7   | 271.1   | 0.650 ms
01:41:56  | 134.8   | 268.3   | 0.642 ms
01:41:57  | 139.3   | 268.3   | 0.629 ms
01:41:59  | 140.9   | 273.9   | 0.639 ms
01:42:00  | 134.8   | 273.9   | 0.575 ms
```

Configuration 設定

在 `config.py` 中

```
"""
Configuration for ECG.
"""

from dataclasses import dataclass

@dataclass
class Config:
    # Serial
    port: str = "/dev/ttyACM0"
    baud_rate: int = 115200
    batch_size: int = 10  # Set one batch (N samples) for each pipeline
                          # 1 loops can process N samples in same batch
    """
    Noted that for the plot frame rate is 30fps,
    which is 0.33s per frame, if batch size too big,
    for ex. 100 samples per batch, then for 500Hz fs,
    100/500 = 200ms, you might notice the plt is lagging
    """

    # Signal
    fs: int = 500        # Sampling frequency (Hz)

    # Filter
    hp_fc: float = 2.0    # Highpass filter cutoff (Baseline Wander)

    # Window lengths
    ma_len: int = 8       # Moving Average Lowpass filter
    deriv_len: int = 8    # Derivative filter
    mwi_len: int = 45     # Moving window integration

    # Peak Detection
    interval: int = 200   # Minimum interval between peaks (ms)
    tau: float = 1.30     # Threshold decay time constant
    min_th: int = 1000    # Minimum Threshold
```

```
# Plot Data buffer
buf_size: int = 2000
```

有詳細的參數設定，要改的 display buffer 是 那個 # Plot Data Buffer 底下的 `buf_size`

其他設定如 Serial port, baud_rate 如上，可以調整為 `COM3` port 之類的

剩下的 Signal processing 設定在中間部份，有詳細註解

檔案架構

- Arduino code 放在資料夾的 `src/` 底下
- Python code 放在 資料夾的 `code/` 底下

Python 版本限制為 3.10 以上，可以用以下指令安裝依賴：

```
pip install -r requirements.txt
```

最後附上 github 連結，裡面的 README.md 有詳細安裝流程：

可以用 `git clone` 下載 repository ：

```
git clone https://github.com/Ohin-Kyuu/py-ecg-monitor.git
```

也有一併附在作業繳交的 .zip 中

> https://github.com/Ohin-Kyuu/py-ecg-monitor