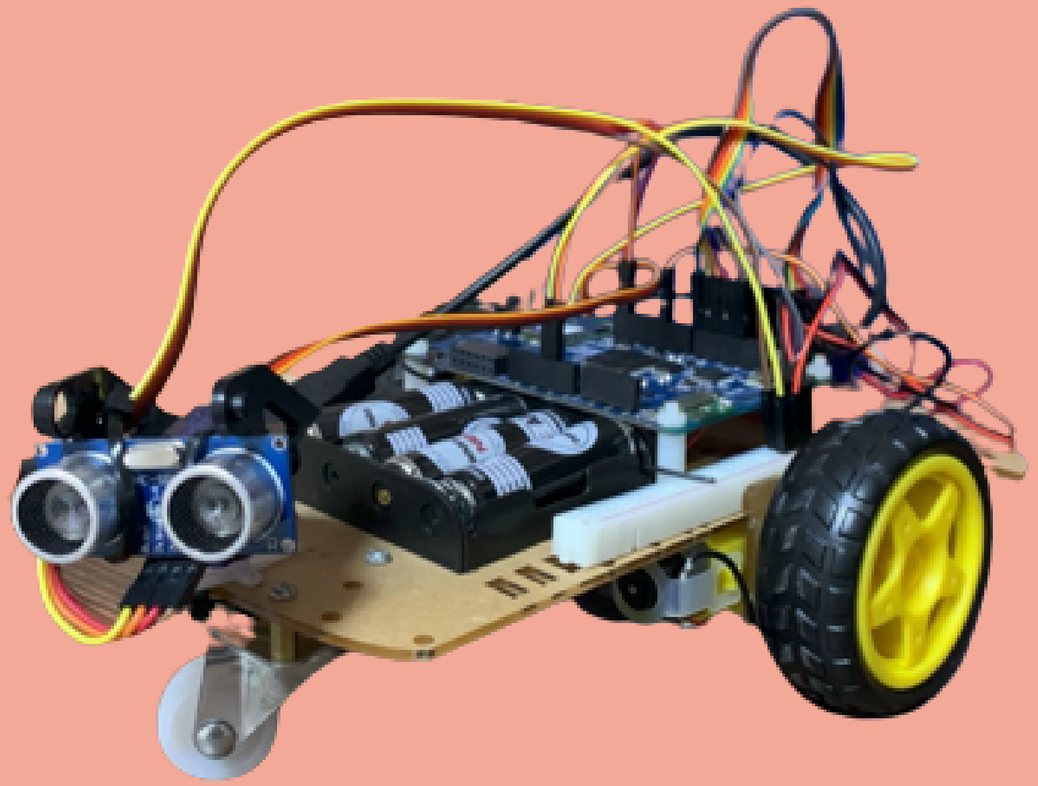
Our IoT Car!

Final Projects report

2022/06/17 鄭至盛、李晨滔

動機：



動機：

由於受到5G物聯網時代、火星探索車與各種多功能的智慧家電影響，本次期末專題選擇透過mbed os的各種功能搭配B-L475E-IOT01A搭載搭載的多樣Sensor與無線通訊模組來實現一台多功能的智慧車。
本次專題中大量使用的Thread、Condition Variable、Mutex、EventQueue與IRQ等功能來提升開發時的效率，並有效整合不同功能，也加入了關於DSP等進階的技巧。

https://github.com/Sam-0403/ESLAB_Final_Project.git

2022/06/17 鄭至盛

專案結構：

專案結構：

CLIENT端

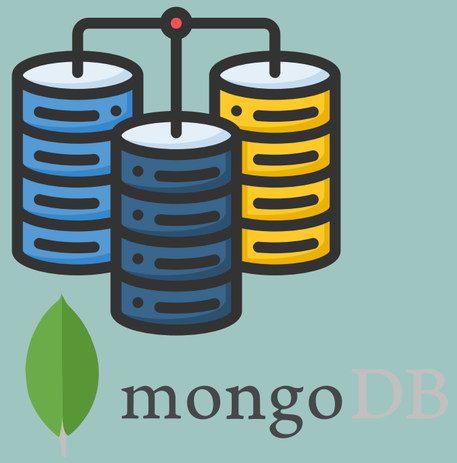
使用PiCamera、OpenCV與Mediapipe
作為手勢辨識的主要Module。

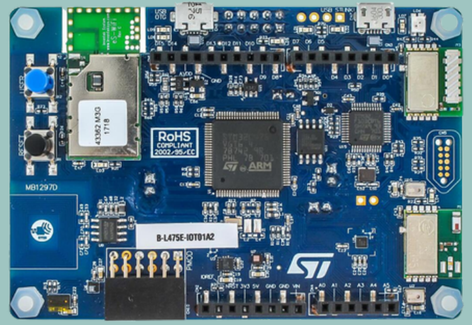
01

SERVER端

串連兩端的Client，接收、發送資訊
並進行對應處理，與資料庫溝通。

02





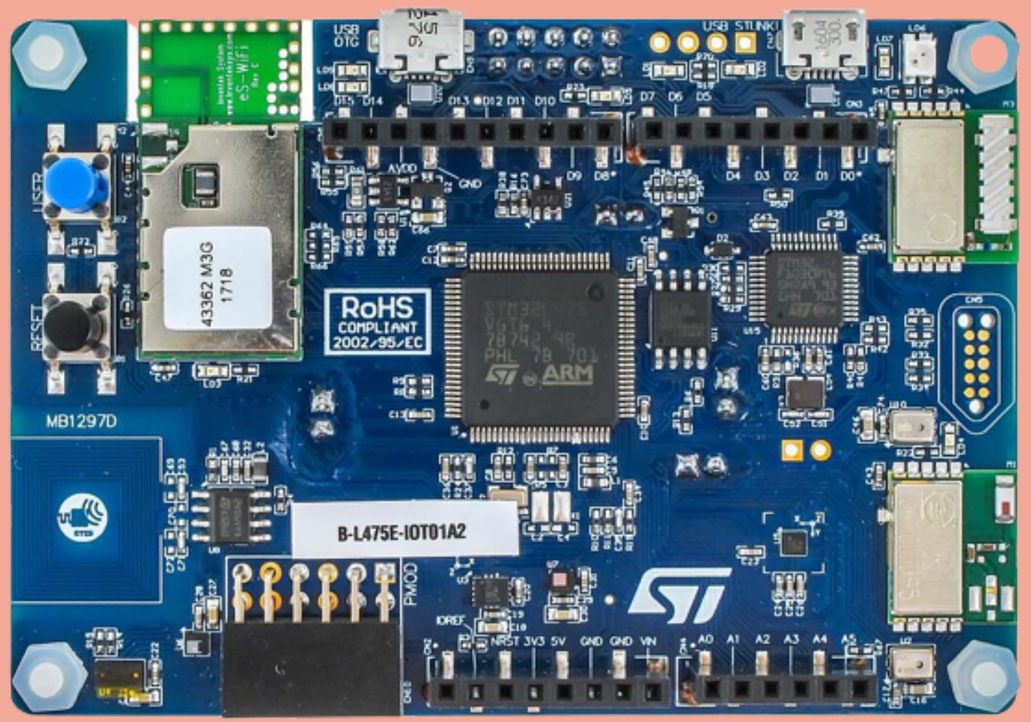
STM32端

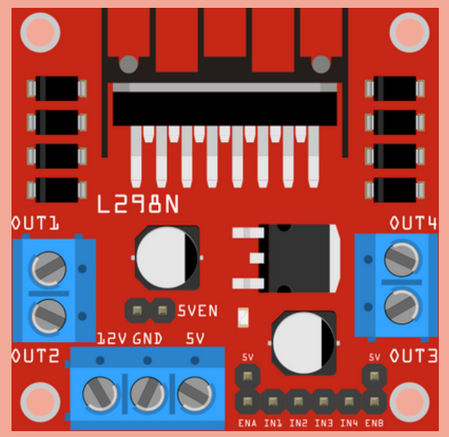
發覺板子上不同功能，透過mbed os
提供的工具進行對應功能設計。

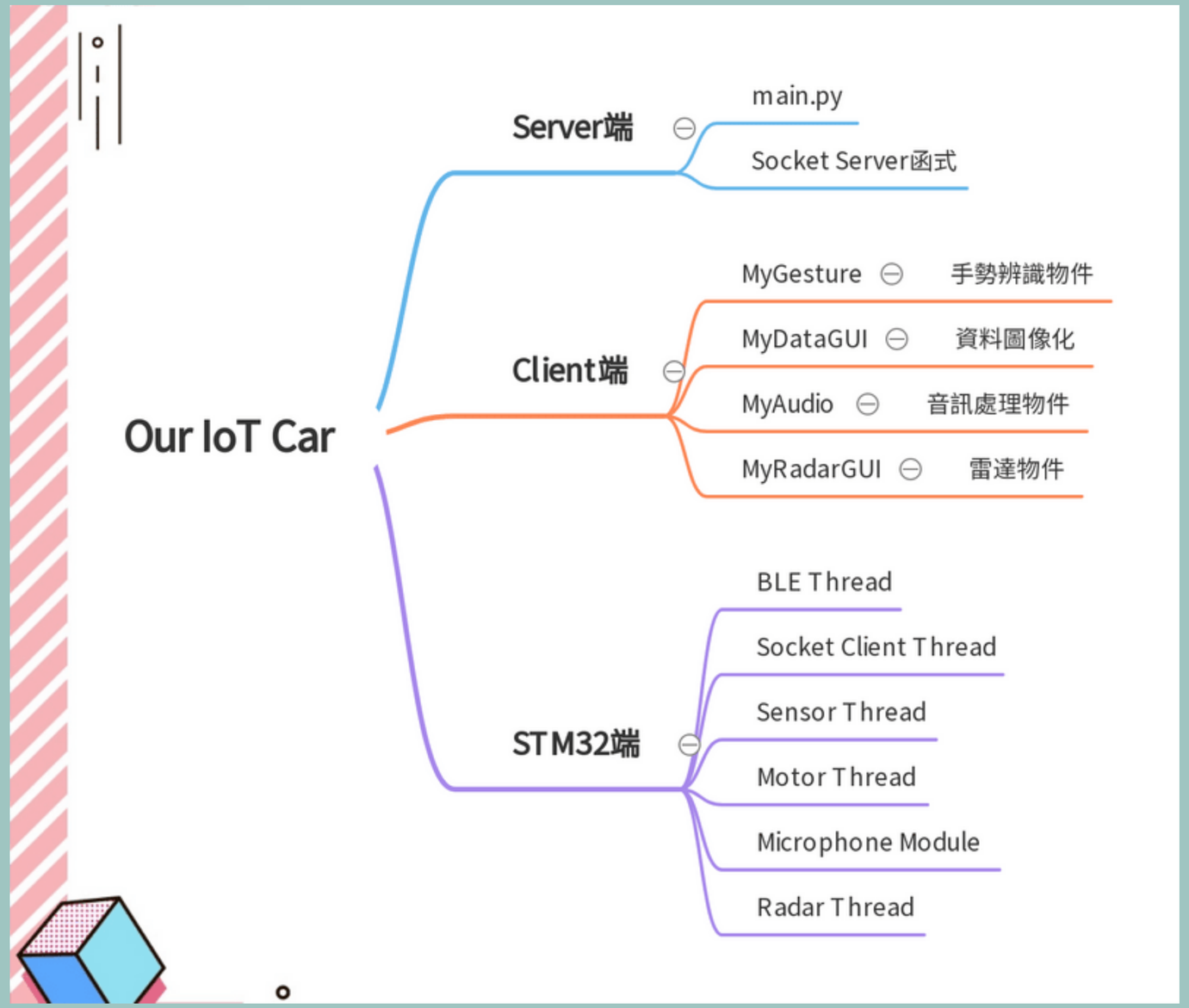
03

2022/06/17 鄭至盛

硬體結構：






Our IoT Car
Server端
main.py
Socket Server函式
Client端
MyGesture
手勢辨識物件
MyDataGUI
資料圖像化
MyAudio
音訊處理物件
MyRadarGUI
雷達物件
STM32端
BLE Thread
Socket Client Thread
Sensor Thread
Motor Thread
Microphone Module
Radar Thread

Mbed端：

SOCKET THREAD

負責網路連線、send_http_request、receive_http_request。

01

RADAR THREAD

05

處理雷達相關的初始化、控制與資料傳輸。

SENSOR THREAD

接收板載sensor的資料，並回傳至server，或進一步處理。

02

MAIN THREAD

07

處理不同Thread間的設定、整合等，並初始化我們需要的物件。

GATT CLIENT THREAD

連線對應的BLE裝置，接收來自GattServer的訊息，並回傳至server。

03

AUDIO THREAD

06

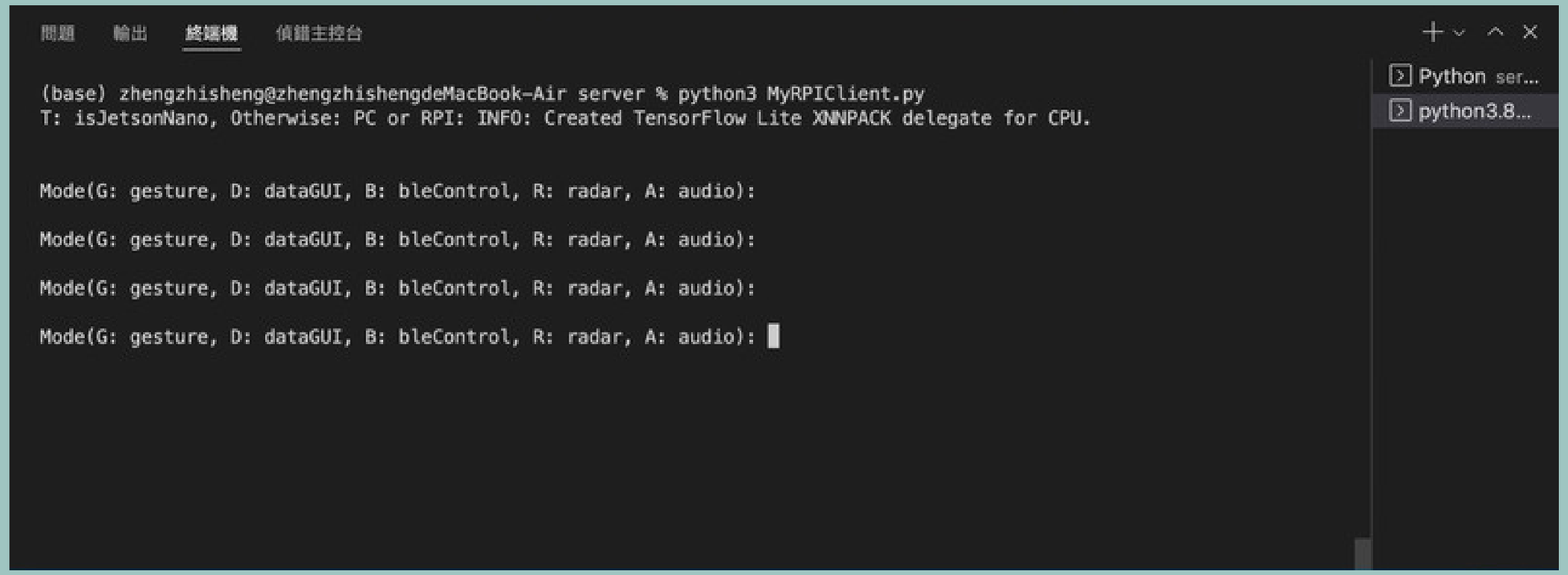
設定音訊對應buffer、處理callback並回傳至server。

MOTOR THREAD

控制L298N需要的pwm、gpio輸出，以控制車車的移動。

04

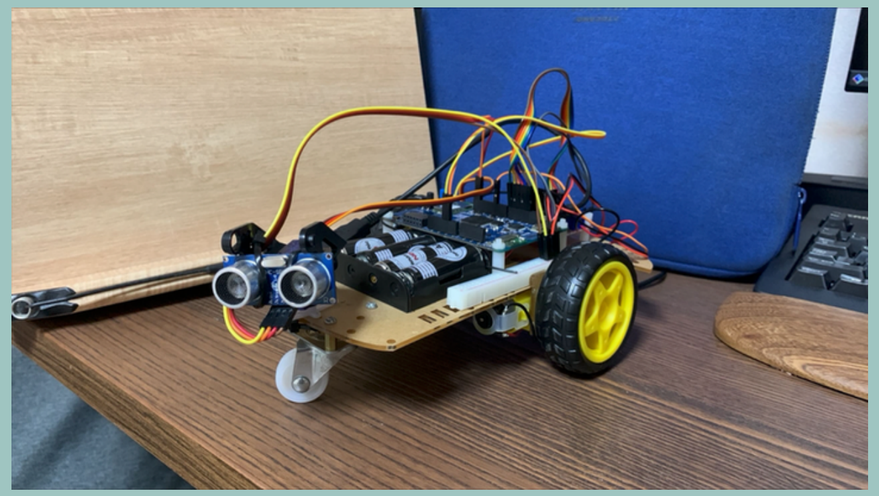
Client端：



透過命令列的形式選擇不同模式

2022/06/17 鄭至盛

不同功能：



Thank Queue

2022/06/17 鄭至盛

main.cpp：

```
EventQueue event_queue;
Semaphore print_sem(1);
Semaphore audio_sem(1);

Mutex mutex_socket;
ConditionVariable cond_socket(mutex_socket);
char socket_buffer[1024];

Mutex mutex_motor;
ConditionVariable cond_motor(mutex_motor);

char* ble_buffer;
bool ble_changed = false;

bool start_radar = false;
char mode = 'N'; // N: unknown, A: audio, R: radar

BLE &ble1 = BLE::Instance();
```

EventQueue負責處理各種IRQ內非同步呼叫函式等與整合不同Thread間的事件

Semaphore負責printf相關事件的控制

cond_socket幫助Thread通知send socket功能

EventQueue
ConditionVariable

不同的Object內均運行專屬的Thread，通過Pass by Reference設定共同的os相關variable

```
mbed_trace_init();
myAudio = new MyAudio(mutex_socket, cond_socket, event_queue, socket_buffer, mode);
 myAudio->start();

myGattClient = new MyGattClient(ble1, event_queue, print_sem, mutex_socket, cond_socket,
socket_buffer, ble_changed, ble_buffer, error_ble, LED1, mode);
 myGattClient->start();

mySocket = new MySocket(print_sem, mutex_socket, cond_socket, socket_buffer,
mutex_motor, cond_motor, motor_state, ble_changed, ble_buffer, LED3, error_socket,
*myGattClient, mode, *myAudio);
 mySocket->start();

 mySensor = new MySensor(mutex_socket, cond_socket, socket_buffer);
 mySensor->start();
myRadar = new MyRadar(mutex_socket, cond_socket, socket_buffer, PA_7, PB_9, PB_8,
mode);
 myRadar->start();

 myMotorControl = new MyMotorControl(0.8, mutex_motor, cond_motor, motor_state);
 myMotorControl->start();
```

2022/06/17 鄭至盛

手勢辦識：

2022/06/17 鄭至盛

Demo影片：

2022/06/17 鄭至盛

功能整理：

✌(G): Go, ✋(S): Stop

(L): Left, (R): Right,
(F): Front, (B): Back

(U): SpeedUp, (D): SlowDown

透過Mediapipe回傳的關節座標點推算手指夾角、手掌方向，通過其對應手勢控制車。

Socket Receive:

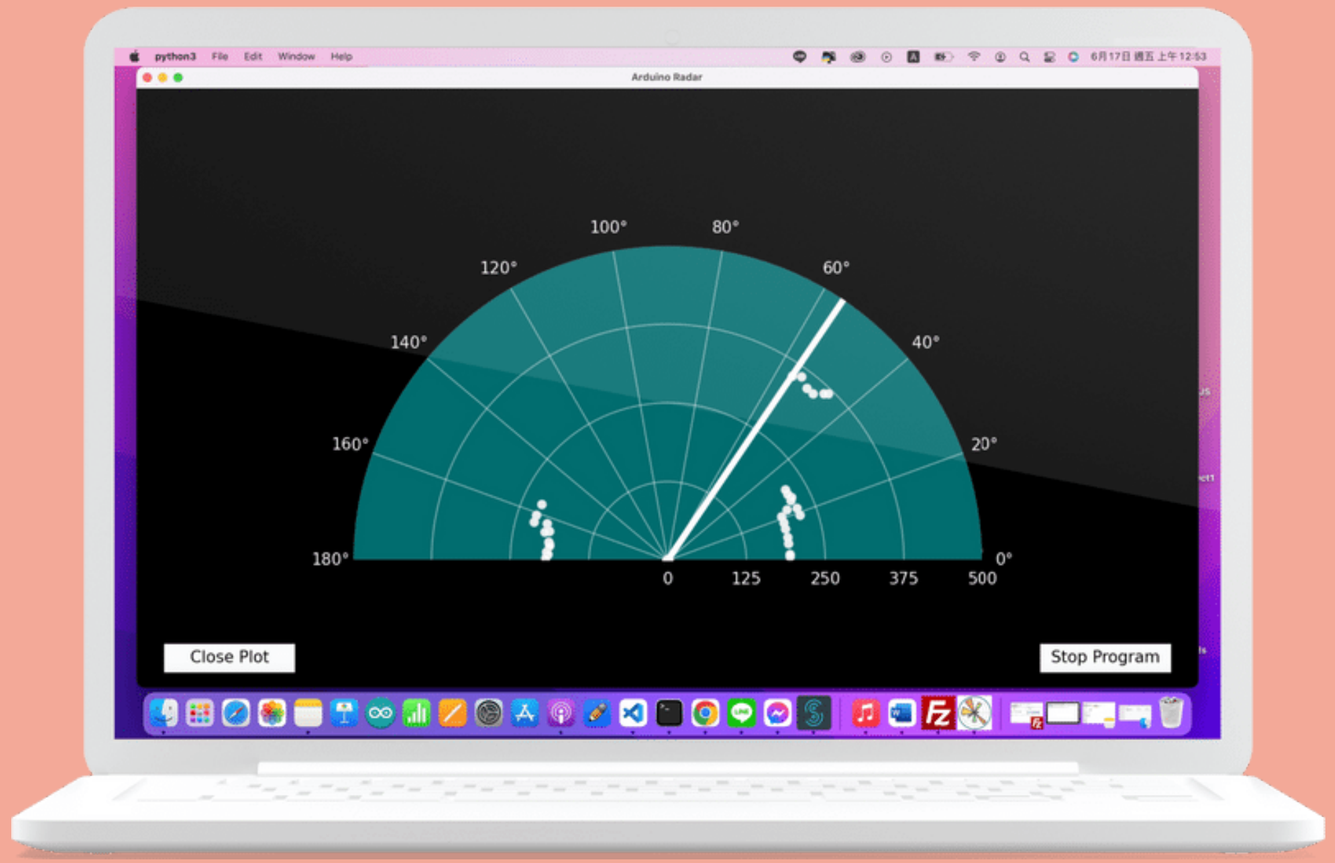
```
if(strstr(buffer, "Go")){
    _mutex_motor->lock();
    *_motor_state = 'G';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "Stop")){
    _mutex_motor->lock();
    *_motor_state = 'S';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "Left")){
    _mutex_motor->lock();
    *_motor_state = 'L';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "Right")){
    _mutex_motor->lock();
    *_motor_state = 'R';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "Front")){
    _mutex_motor->lock();
    *_motor_state = 'F';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
```

透過cond_motor來同步呼叫Motor Thread調整motor_state，以切換到對應功能。

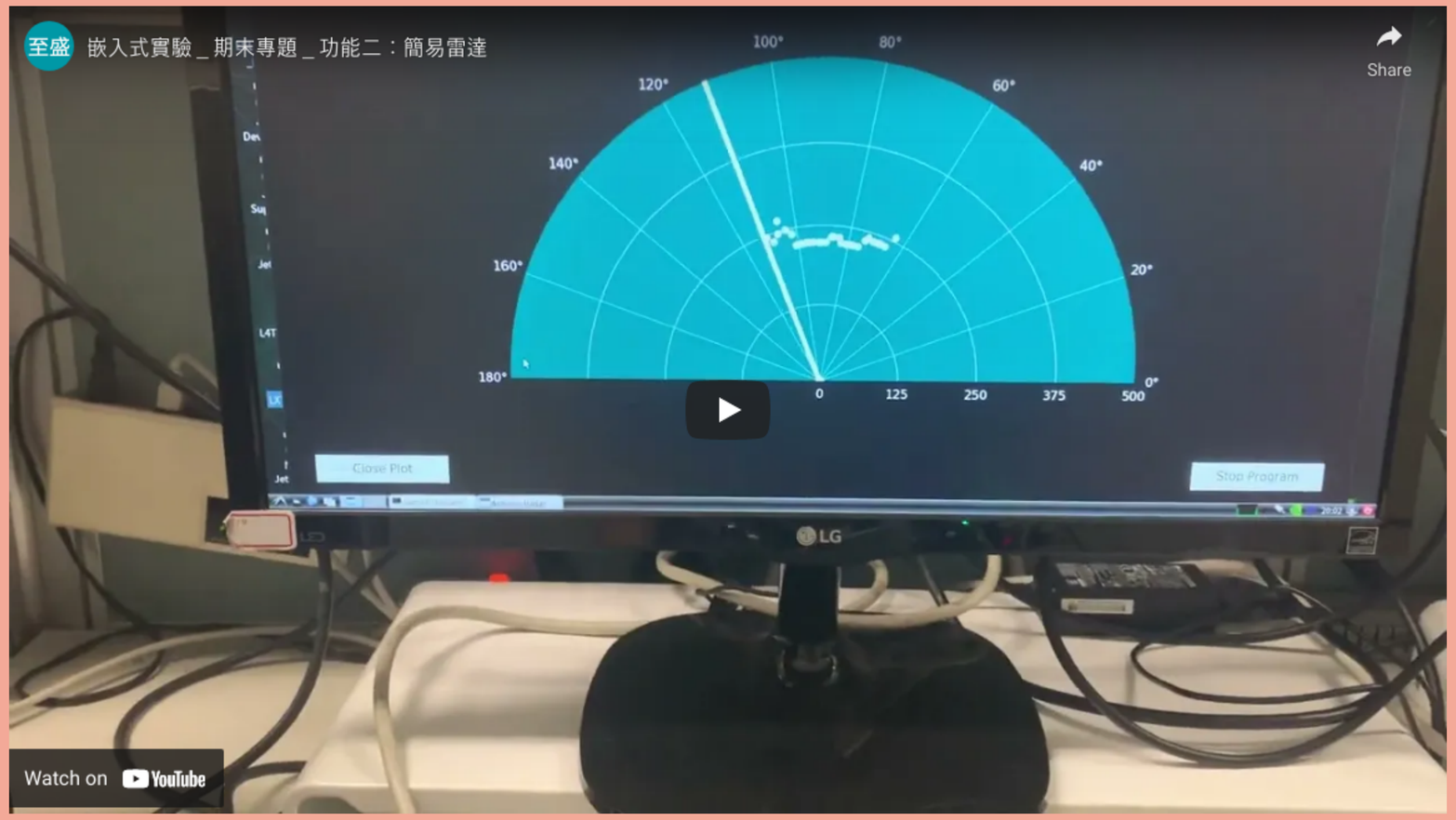
但此方法無法針對BLE與Audio相關Thread使用，因為其大多透過callback來呼叫對應函式，無法如同Motor Thread使用loop，否則會出現callback與loop衝突，造成Fault Error。

2022/06/17 鄭至盛

超音波相關：



Demo影片：


至盛
嵌入式實驗 _ 期末專題 _ 功能二：簡易雷達
Share
100°
80°
120°
60°
140°
40°
160°
20°
180°
0°
0
125
250
375
500
Close Plot
Stop Program
LG
Watch on YouTube

實作方法：

初始化SG90伺服馬達物件

初始化HC_SR04偵測器物件

通過設定PWM的佔空比設定馬達角度

當偵測器偵測到回傳聲波時通過時間計算障礙物距離（為一IRQ事件）

通過Condition Variable呼叫socket

遇到問題：

```
SG90::SG90(PinName pwmPin){
    iPreRotate=1450;
    Pin = new PwmOut(pwmPin);
    // period_ms(20);// 50Hz to trigger SG90
    Pin->period_ms(20);
    // pulsewidth_us(iPreRotate); //500:90 1450:0 2400:-90
    Pin->pulsewidth_us(iPreRotate);
}
void SG90::SetAngle(float fAngle){
    if (fAngle>90.0f)
        fAngle=90.0f;
    else if(fAngle<-90.0f)
        fAngle=-90.0f;
    int iRotate=(-(fAngle-90.0f)*950.0f/90.0f)+500.0f;
    // pulsewidth_us(iRotate);
    Pin->pulsewidth_us(iRotate);
}
```

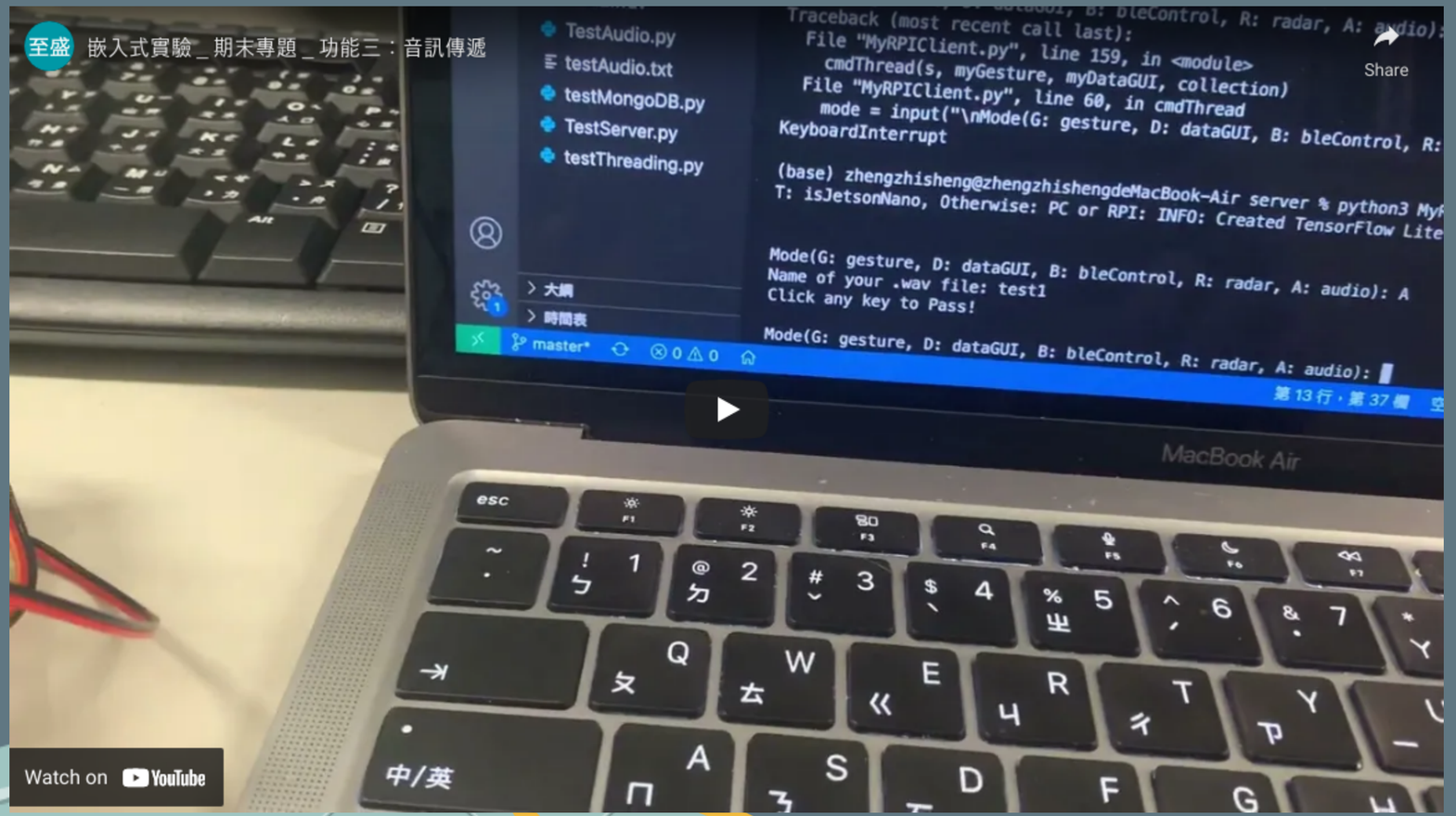
原始的SG90採用Inheritance
的方式，繼承PwmOut。
但會造成PwmOut與SG90的Timer衝突
我沒有很確定成因為何，但修改為獨立的
class即可避免此問題。

2022/06/17 鄭至盛

音訊相關：

2022/06/17 鄭至盛

Demo影片：



2022/06/17 鄭至盛

實作方法：

引入對應HAL庫（MP34DT01）

初始化MyAudio物件，設定對應callback

設定buffer，儲存音訊資料並切割成Partition
通知socket傳輸（與設定wav header）

client端將資料寫為wav檔並播放出來

2022/06/17 鄭至盛

問題統整：

1.過高的sample rate

由於無法單純修改sample rate，改為透過原sample進行取樣。（Downsampling）

2.傳輸效率不足

由於傳輸效率不足，實際的callback會被傳輸速率限制導致實際取樣時轉換的聲音會有加速的感覺。

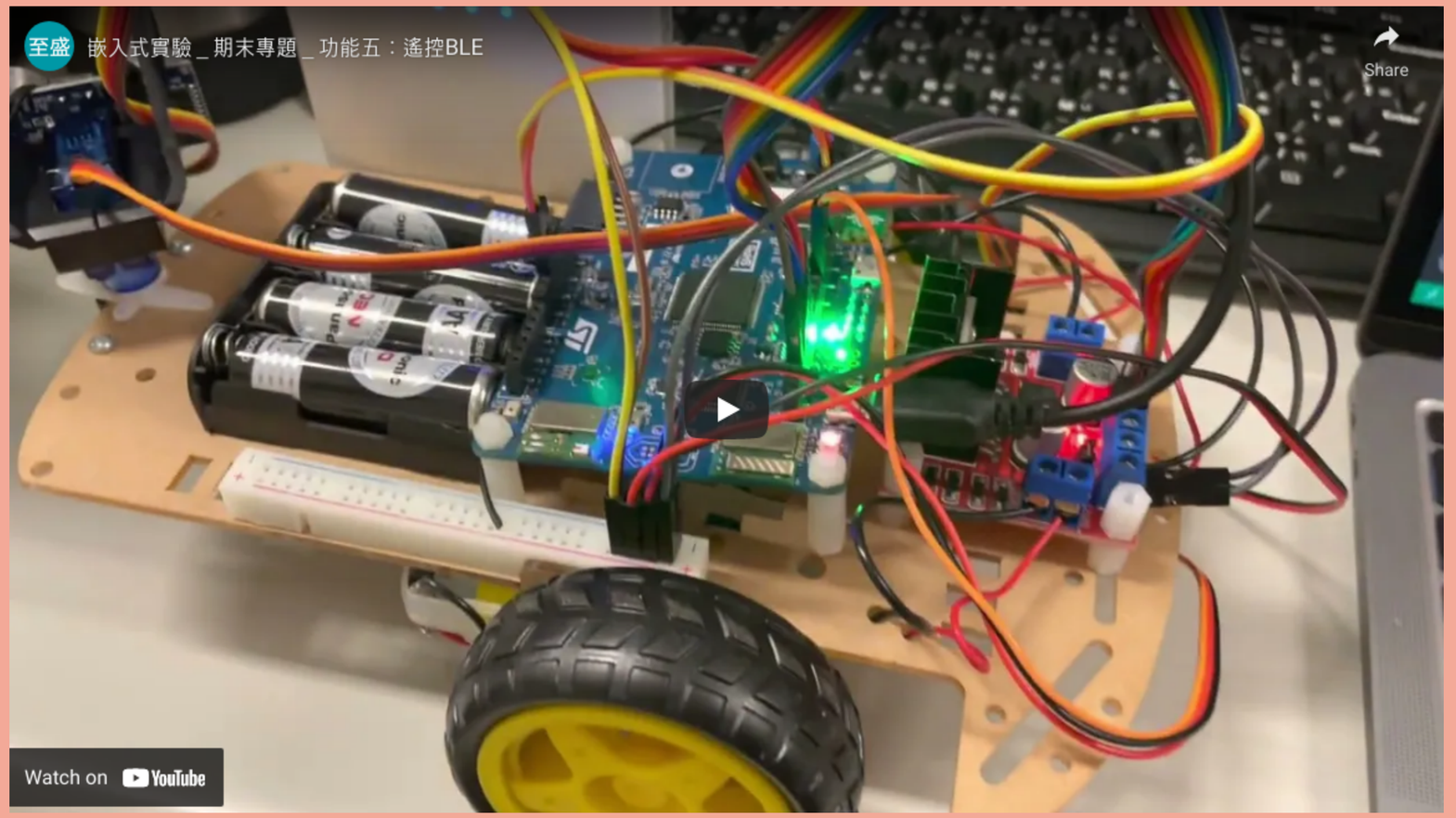
3.記憶體容量不足

由於記憶體不足，每次callback最多只能為0.25秒。造成聲音會容易出現斷續的現象。

2022/06/17 鄭至盛

BLE相關：

Demo影片：



實作方法：

初始化ble相關物件

伺服器傳送連線裝置名稱

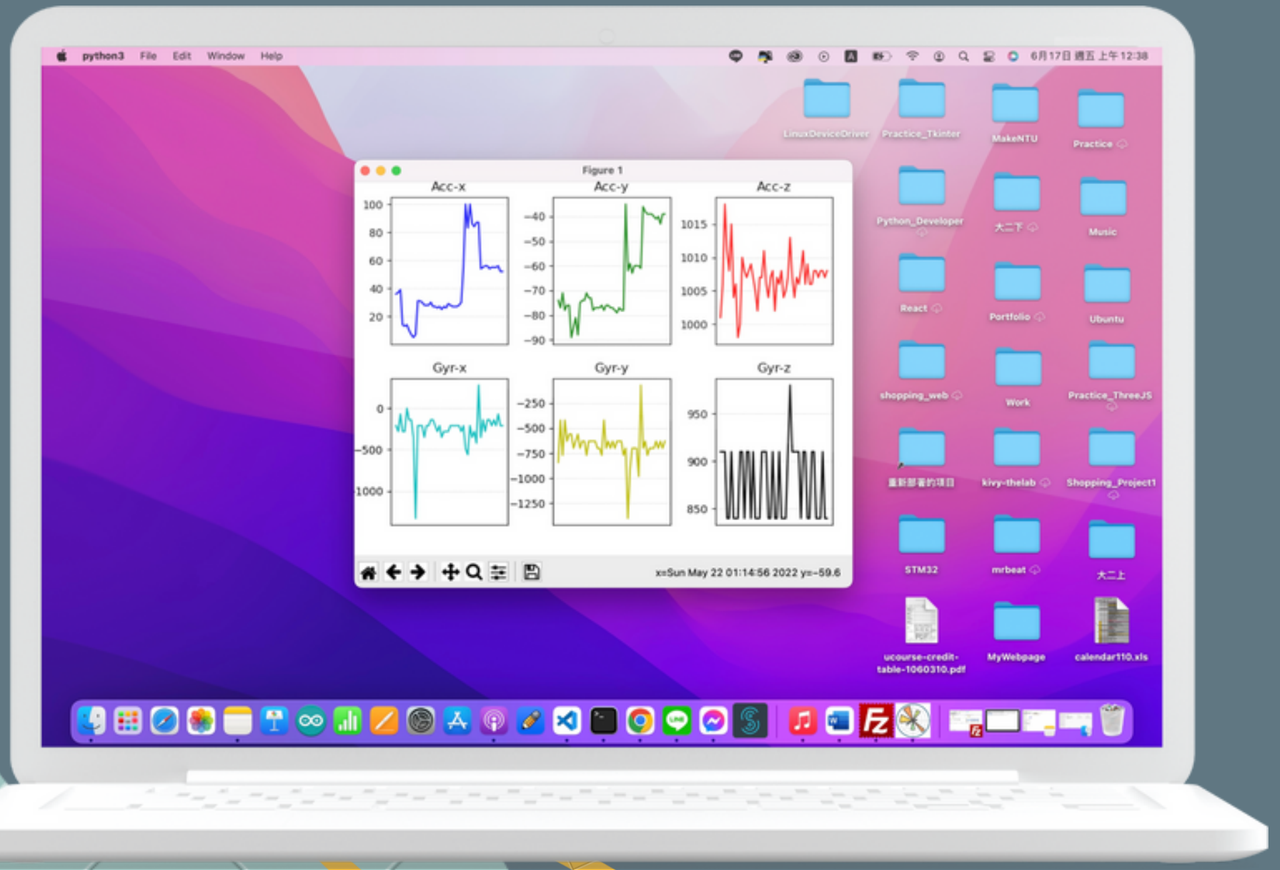
socket Thread在接受時呼叫BLE連線函式

當偵測到characteristic改變時，呼叫socket相關condition variable以傳送資料

曾嘗試為ble設定對應condition variable，但會出現無法解決的錯誤

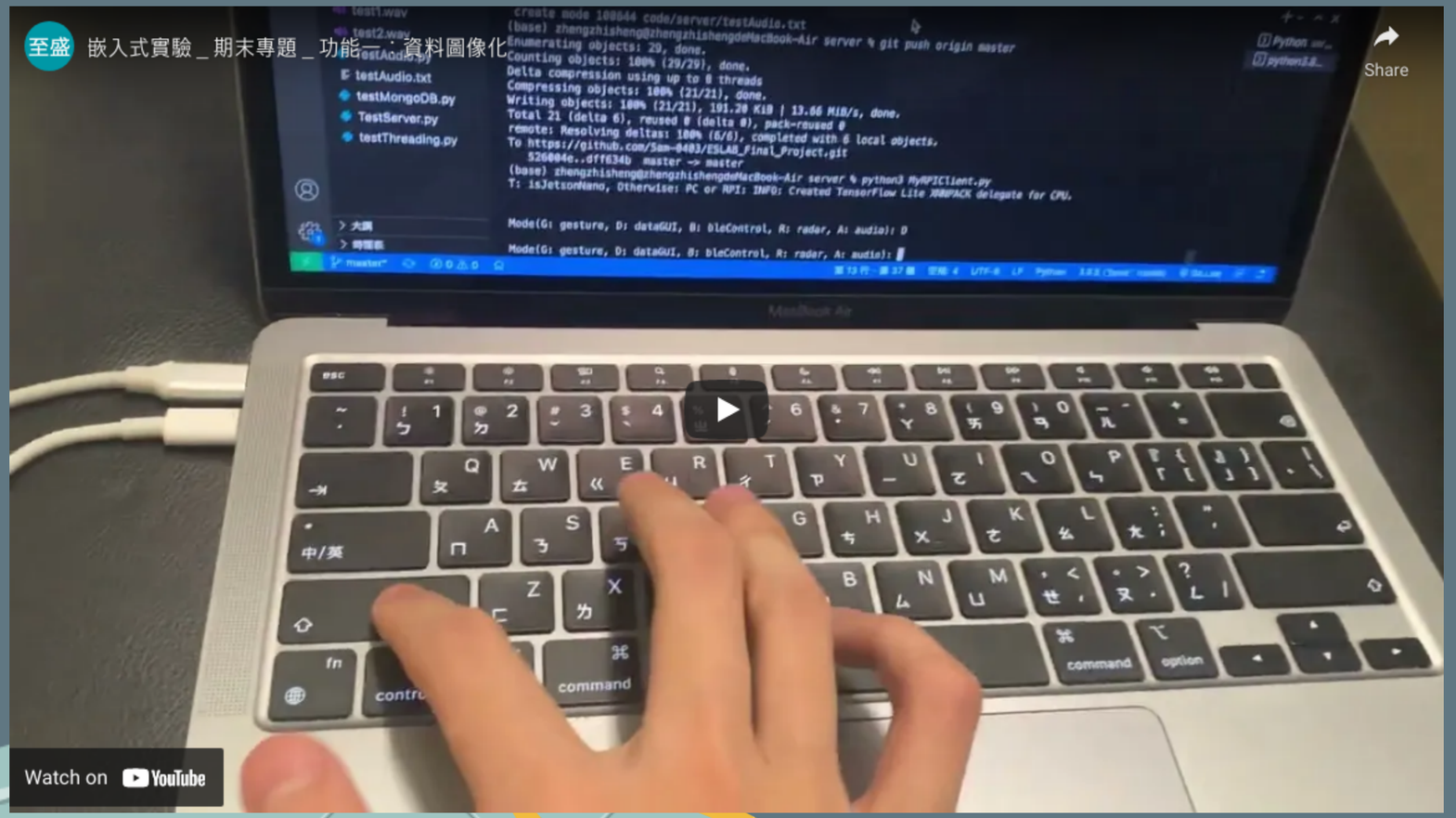
2022/06/17 鄭至盛

資料圖像化：



2022/06/17 鄭至盛

Demo影片：



2022/06/17 鄭至盛

實作方法：

伺服器將sensor資料傳入資料庫

啟用資料圖像化功能時至資料庫呼叫近50筆資料

依據資料順序（也可以是時間）做圖

透過server傳送的資料實時更新資料

mongoDB

參考資料

2022/06/17 鄭至盛

Jetson Nano設定：

free -h　//查看當前swap狀態

sudo fallocate -l 4G /var/swapfile
sudo chmod 600 /var/swapfile
sudo mkswap /var/swapfile
sudo swapon /var/swapfile
sudo bash -c 'echo "/var/swapfile swap swap defaults 0 0" >> /etc/fstab'

cv2.VideoCapture('nvarguscamerasrc ! nvvidconv ! video/x-raw, format=BGRx ! videoconvert ! video/x-raw, format=BGR ! appsink', cv2.CAP_GSTREAMER)

Wav Header設定：

Name	offset	Size	Value
ChunkID	0	4	“RIFF”
ChunkSize	4	4	405040
Format	8	4	“WAVE”
Subchunk1 ID	12	4	“fmt”
Subchunk1 Size	16	4	16
Audio Format	20	2	1
Num Channels	22	2	2
Sample Rate	24	4	22050
Byte Rate	28	4	88200
Block Align	32	2	4
Bits per Sample	34	2	16
Subchunk2 ID	36	4	“data”
Subchunk 2 Size	40	4	405004

```
unsigned wav_header[44] = {
 0x52, 0x49, 0x46, 0x46, // RIFF
 fileSize & 0xff, (fileSize >> 8) & 0xff, (fileSize >> 16) & 0xff, (fileSize >> 24) & 0xff,
 0x57, 0x41, 0x56, 0x45, // WAVE
 0x66, 0x6d, 0x74, 0x20, // fmt
 0x10, 0x00, 0x00, 0x00, // length of format data
 0x01, 0x00, // type of format (1=PCM)
 0x01, 0x00, // number of channels
 wavFreq & 0xff, (wavFreq >> 8) & 0xff, (wavFreq >> 16) & 0xff, (wavFreq >> 24) &
0xff,
 sbcHeader & 0xff, (sbcHeader >> 8) & 0xff, (sbcHeader >> 16) & 0xff, (sbcHeader >>
24) & 0xff, // (Sample Rate * BitsPerSample * Channels) / 8
 0x02, 0x00, 0x10, 0x00,
 0x64, 0x61, 0x74, 0x61, // data
 dataSize & 0xff, (dataSize >> 8) & 0xff, (dataSize >> 16) & 0xff, (dataSize >> 24) &
0xff,
 };
```

2022/06/17 李晨滔

後記：DSP

$$\frac{Y(e^{j\omega})}{X(e^{j\omega})} = \frac{\sum_{k=0}^{M} b_k e^{-jk\omega}}{\sum_{k=0}^{N} a_k e^{-jk\omega}} \leftrightarrow \sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

```
static double b_0 = 0.0008;
static double b_1 = 0.0024;
static double b_2 = 0.0024;
static double b_3 = 0.0008;

static double a_0 = 1;
static double a_1 = -2.6079;
static double a_2 = 2.2890;
static double a_3 = -0.6748;
```

實作方法

- 3-order Butterworth lowpass filter
- MATLAB: butter(3,fc/(fs/2))
  where fc = 500Hz, fs = 16kHz

- human voice frequency range
- typical adult male: 85~180Hz
  typical adult female: 165~255Hz

遭遇到的問題：記憶體不足

```
static int16_t *TARGET_AUDIO_BUFFER = (int16_t*)calloc(TARGET_AUDIO_BUFFER_NB_SAMPLES, sizeof(int16_t));
static int16_t *TEMP_TARGET_AUDIO_BUFFER = (int16_t*)calloc(TARGET_AUDIO_BUFFER_NB_SAMPLES, sizeof(int16_t));
```

```
int16_t *buf_16t = TARGET_AUDIO_BUFFER;
int16_t *buff_temp = TEMP_TARGET_AUDIO_BUFFER;

    for (int16_t ix = 0; ix < TARGET_AUDIO_BUFFER_NB_SAMPLES; ix++) {
        if (ix >= 3) {
            double tmp_output =(b_0*buf_16t[ix] + b_1*buf_16t[ix-1] + b_2*buf_16t[ix-2] + b_3*buf_16t[ix-3])-(a_1*buff_temp[ix-1] - a_2*buff_temp[ix-2] - a_3*buff_temp[ix-3]);
            buff_temp[ix] = (int16_t)tmp_output;
        }
        else {
            buff_temp[ix] = buf_16t[ix];
        }
    }
```

input buffer
output buffer
ヲシト容量不足!!
(previous) input
previous output

1. Originally, sampling frequency = 16kHz
若只錄1秒→需要16000*2*2=64000bytes
若錄2秒→128000bytes!在僅考慮audio recording的任務時就已經跳HardFault error

2. Trade-off: recursive filter實作容易，但可能需要較多記憶體；然而直接做convolution需要較大量的運算!

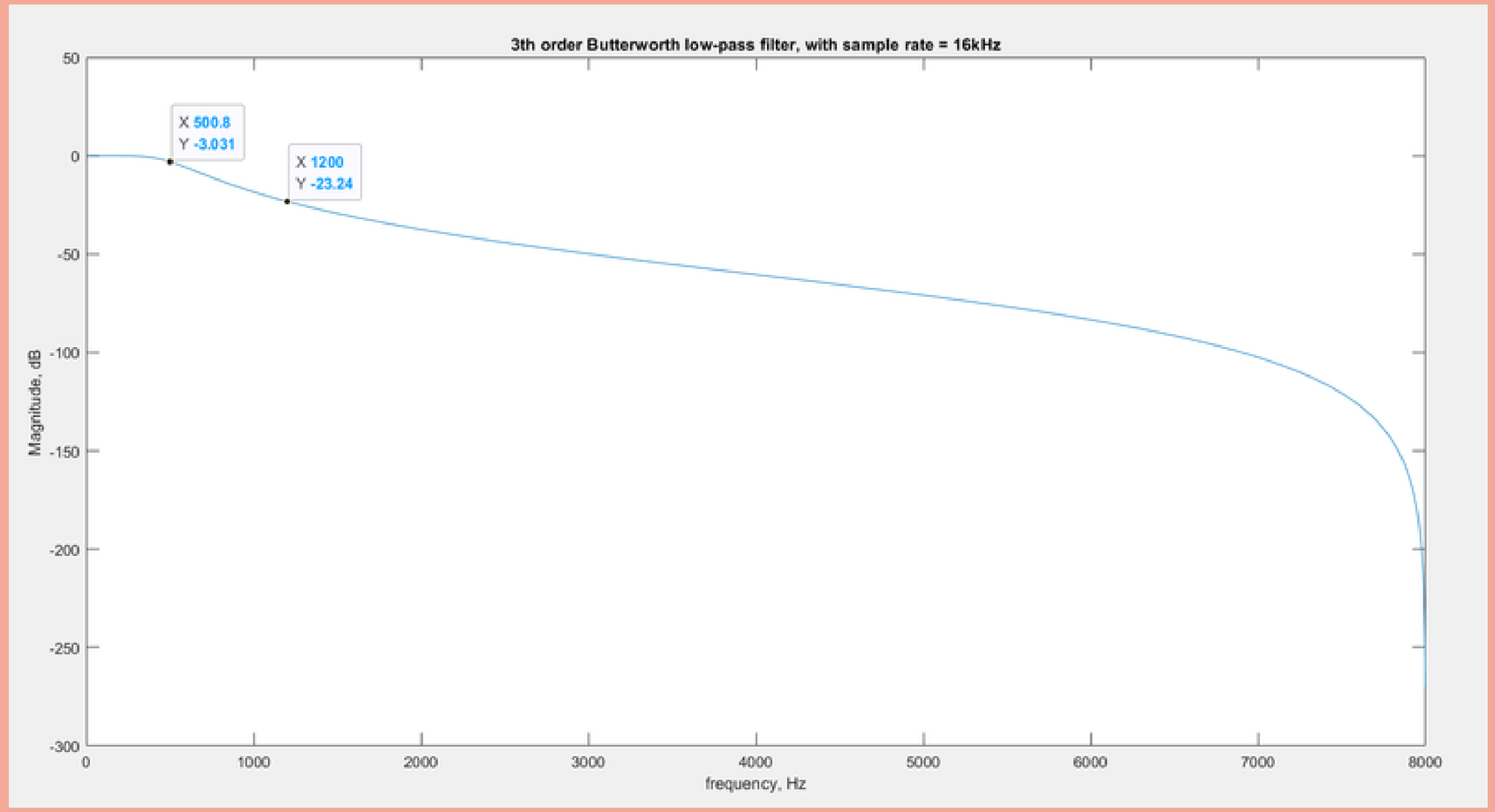
參考資料:https://github.com/janjongboom/b-l475e-iot01a-audio-mbed

2022/06/17 李晨滔

目前DSP的結果

1 錄製一秒，使用人聲(我)+單一高頻音(1200Hz)做為input

2 Bode plot(Magnitude)



3 成果展示

Q&A Time!