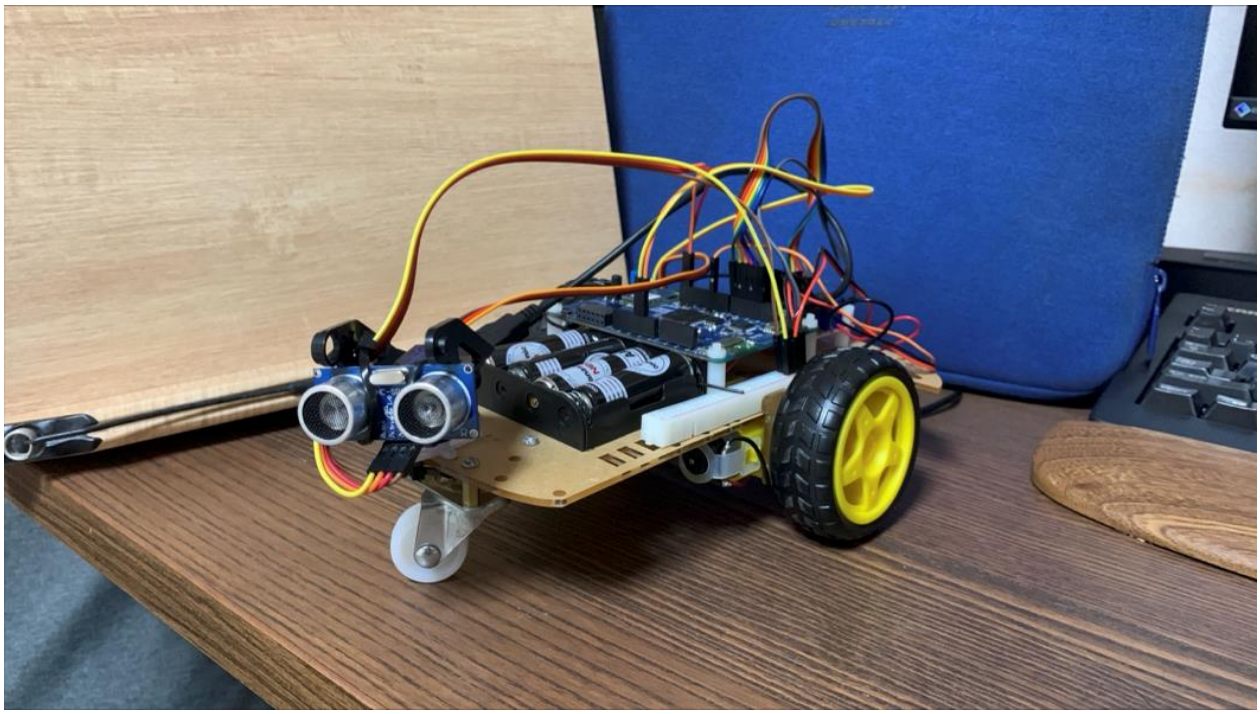
ESLab：2022Spring－Final

嵌入式期末專案：多功能智慧車

B09901095/電機二/鄭至盛
B06703027/電機四/李晨滔



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

Demo Videos：

1. 功能一：資料圖像化: https://www.youtube.com/watch?v=Mk69kflESu8
2. 功能二：簡易雷達: https://www.youtube.com/watch?v=RHm4_1YgY9E
3. 功能三：音訊傳遞: https://www.youtube.com/watch?app=desktop&v=UVQB6Vfc58c&feature=youtu.be
4. 功能四：手勢辨識遙控: https://www.youtube.com/watch?v=I0MvmhiXyvU
5. 功能五：遙控 BLE: https://www.youtube.com/watch?v=5mQlsj9Lum0

DSP results：
https://drive.google.com/drive/folders/1k4PmEDyh50bGFpKONWslem57t9Nv4jO0?usp=sharing
(由於 wav 檔無法上傳到 Youtube，故我們改上傳到 Google Drive，再請助教去雲端上下載了。)

動機：

(1) 做整合式專案的動機：

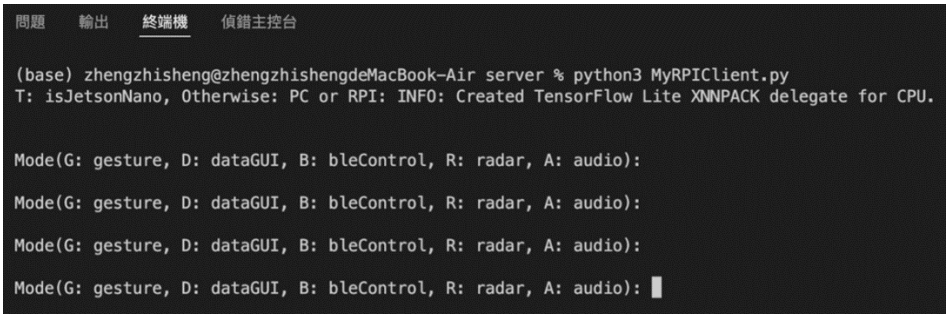
由於受到 5G 物聯網時代、火星探索車與各種多功能的智慧家電影響，本次期末專案選擇透過 Mbed Os 的各種功能搭配 B-L475E-IOT01A 搭載的多樣 sensor 與無線通訊模組來實現一台多功能的智慧車。

本項期末專案中，通過建立 Server、Client、Database 與控制板的 Socket 連線來模擬遠端遙控智慧車的功能。此外，本次專案中大量使用 Mbed OS 支援的 Thread、Condition Variable、Mutex、EventQueue 與 IRQ 等作業系統相關的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。(幾乎所有功能都有使用到上述作業系統相關的功能)

(2) 做 DSP 的動機：

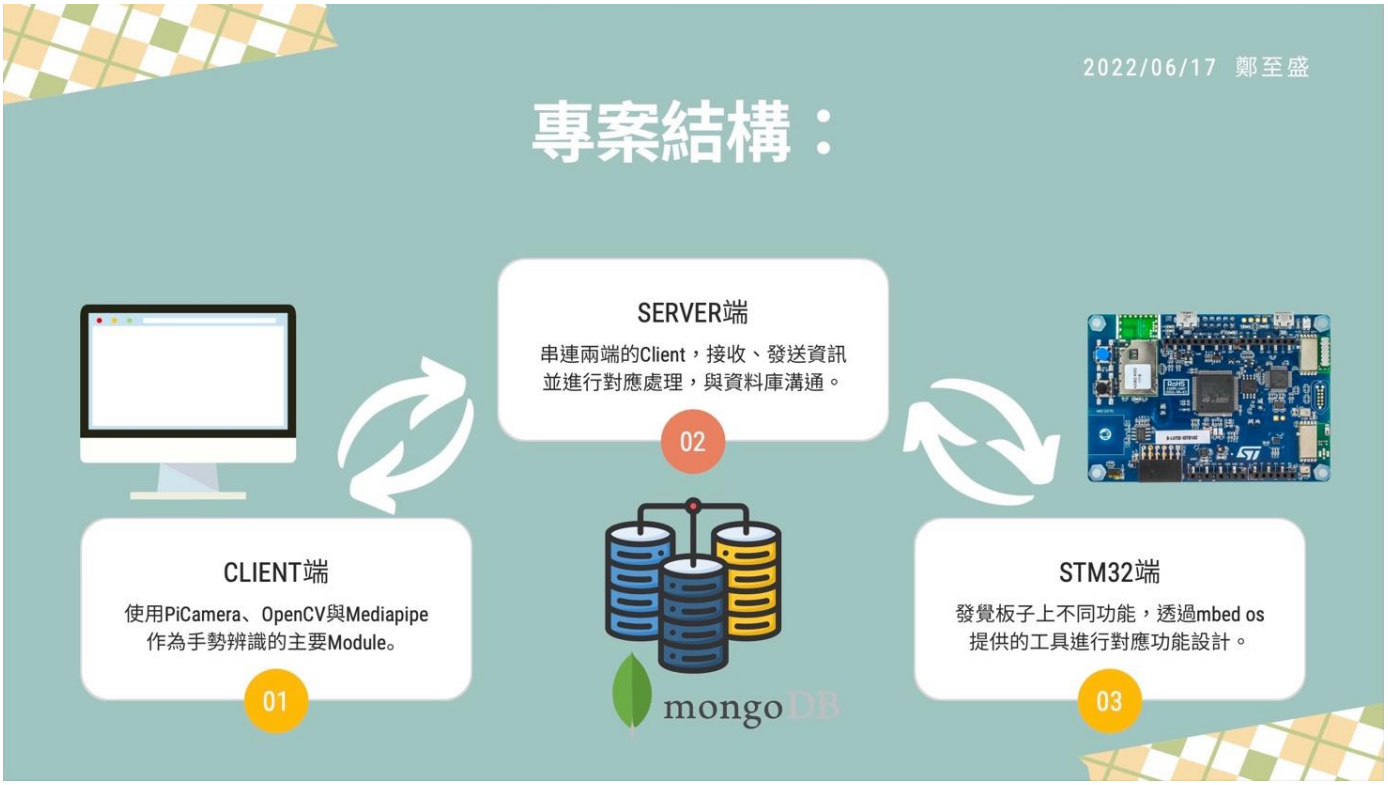
數位訊號處理(Digital Signal Processing)一直都是電機工程領域中相當重要的應用，舉凡一般之通訊、無線電波傳輸、語音辨識等等都會使用到 DSP 的概念，而我們在著手本次期末專案時，研究了一段時間後發現 STM32 上其實有麥克風可以使用，便開始上網查詢相關的模組、套件。後來成功將查詢到的套件和模組移植到 Mbed OS，我們就進一步思考，或許可以對收到的音訊進行 DSP，如此一來不僅有機會可以和其他專案做結合，也可以用實作來驗證我們在信號與系統以及嵌入式實驗課程中所學到的 DSP 知識。

專案概述：



（↑用命令列輸入想要執行的功能，可選擇輸入 G、D、B、R 和 A）

B09901095/電機二/鄭至盛、B06703027/電機四/李晨滔



B09901095/電機二/鄭至盛、B06703027/電機四/李晨滔

作法：

Mbed OS API 設定：

```
EventQueue event_queue ;
// 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 print_sem(1) ;
Semaphore audio_sem(1) ;
// Semaphore 負責 printf 相關事件的控制並控制有限的資源

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

Mutex mutex_motor ;
ConditionVariable cond_motor(mutex_motor) ;
```

一些在 main.cpp 中，與上課講過的內容有關的程式碼:

```
char* ble_buffer;
bool ble_changed = false;
BLE &ble1 = BLE::Instance();
```

1. 功能一：資料圖像化

在 Mbed 端，當 Sensor Thread 定期接收 sensor 資料時會將對應資料寫入 socket_buffer 中，並通知 cond_socket，此時 Socket Thread 會將資料傳送出去，通過 Server 將資料記錄至 Database 中。

在 Client 端，透過呼叫 Database 我們可以獲得當下和以前的資料，而此時 Server 也會實時地更新 Client 端的資料，通過 Matplotlib 將資料圖像化，顯示車子在當前環境感應的各種數據。

2. 功能二：簡易雷達

在 Mbed 端，通過初始化 SG90 與 HC-SR04 物件來接收資料、控制伺服馬達等，而本次由於我們的功能較複雜，無法單純地使用我們查到的函式庫，而需要在 callback 或結構等地方進行對應地修改。當 Socket Thread 接收到對應訊號時會呼叫 MyRadar，在 MyRadar 中通過將資料寫入 socket_buffer 中並通知 cond_socket 來傳送對應資料，而當不使用時也可以停止這些功能以提升效能。

在 Client 端，通過 Matplotlib 可以有效地將當前雷達數據顯示出來，並在有需要時再呼叫此功能以降低控制板端的負載。

此外，也通過設定 PWM 的佔空比來設定馬達角度；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）；也通過 Condition Variable 呼叫 socket。

以下是功能二之專案中使用到的一些上課教過的內容(並無全部列出，僅列出一些代表性的):
在 MyRadar.cpp 中:

```
MyRadar::MyRadar(Mutex &mutex, ConditionVariable &cond, char* buffer, PinName pinServo, PinName pinTrig, PinName pinEcho, char &mode):
    _mutex(&mutex), _cond(&cond), _buffer(buffer), servo(pinServo), radar(pinTrig, pinEcho, 0.1, 1, dist), _mode(&mode){
    // radar(pinTrig, pinEcho, 0.1, 1, &MyRadar::onRadarUpdate);
    // radar(pinTrig, pinEcho, 0.1, 1, dist);
}
```

```
14  void MyRadar::start(){
15      _threadServo.start(callback(this, &MyRadar::setupServo));
16      _threadRadar.start(callback(this, &MyRadar::setupRadar));
17  }
```

```
40          ThisThread::sleep_for(60ms);
```

```
49          if(*_mode=='R'){
50              if(radar.checkDistance()){
51                  _mutex->lock();
52                  // printf("Angle: %f, Dist: %d\n", angle+90, dist);
53                  int len = sprintf(_buffer, "{\"type\":\"STM_RADAR\", \"Angle\":%f, \"Dist\":%d, \"dest\":\"%s\"}", angle+90, dist, DEST);
54                  _cond->notify_all();
55                  _mutex->unlock();
56              }
57          }
58          ThisThread::sleep_for(100ms);
```

3. 功能三：音訊傳遞

在 Mbed 端，通過控制板 HAL 庫提供的 stm32l475e_iot01_audio.h 來設定對應的 callback，此外，由於預設的 sample rate 為 16000Hz，在資料儲存與傳輸方面會產生許多問題，我們也嘗試修改預設 macro，但實際錄到的資料會出現異常，因此改為透過 down sampling 的方法來取得我們需要的傳輸資料（8000Hz），並通過設定 Partition 來傳輸，降低無謂的 Server buffer 空間，但目前實際傳輸的速率仍然不夠，且傳輸時間會造成音訊不連續（或許可以透過 eventqueue.call 解決），因此會造成 Client 端的資料出現快進與不連續。此外，根據當前狀態我們也須設定對應的 wav header，通過設定對應 sample rate、file size、data size 等來完成正確地可以播放地.wav 檔。

在 Client 端，讓使用者自己設定想要錄音的時間，在錄音中實時地更新 audio data，並在錄音結束後通過 write byte 將資料轉換為.wav 檔並播放。

以下是功能三之專案中使用到的一些上課教過的內容(並無全部列出，僅列出一些代表性的):
在 MyAudio.cpp 中:

```
7   MyAudio::MyAudio(Mutex &mutex, ConditionVariable &cond, EventQueue &event_queue, char* buffer, char &mode):
8       _mutex(&mutex), _cond(&cond), _event_queue(&event_queue), _buffer(buffer), _mode(&mode){
9
10  }
```

```
55      for (size_t i=0; i<NB_PARTITION; i++){
56          _mutex->lock();
57          len = sprintf(_buffer, "{\"type\":\"STM_AUDIO\", ");
58          len += sprintf(_buffer+len, "\"data\":\"");
59          for (size_t j=0; j<SIZE_OF_PARTITION/2; j++) {
60              len += sprintf(_buffer+len, "%02x", buf[ix]);
61              len += sprintf(_buffer+len, "%02x", buf[ix+1]);
62              ix+=shift;
63          }
64          len += sprintf(_buffer+len, "\", \"dest\":\"%s\"}", DEST);
65          _cond->notify_all();
66          _mutex->unlock();
67          // ThisThread::sleep_for(2ms);
68      }
```

```
97          _mutex->lock();
98          len = sprintf(_buffer, "{\"type\":\"STM_HEADER\", ");
99          len += sprintf(_buffer+len, "\"data\":\"");
100         for (size_t ix = 0; ix < 44; ix++) {
101             len += sprintf(_buffer+len, "%02x", wav_header[ix]);
102         }
103         len += sprintf(_buffer+len, "\", \"dest\":\"%s\"}", DEST);
104         _cond->notify_all();
105         _mutex->unlock();
106     }
```

```
205     if (TARGET_AUDIO_BUFFER_IX >= TARGET_AUDIO_BUFFER_NB_SAMPLES && isRecording) {
206         _event_queue->call(callback(this, &MyAudio::print_stats));
207         _event_queue->call(callback(this, &MyAudio::target_audio_buffer_full));
208         return;
209     }
210 }
```

```
357 void MyAudio::start(){
358     _thread.start(callback(this, &MyAudio::setup));
359 }
```

4. 功能四：手勢辨識遙控

在 Client 端，通過 Mediapipe 取得手勢辨識地關節座標，並透過計算的方式定義手指角度、手掌方向等推算手勢，在對應的手勢傳送對應地控制訊號至車子。此外，由於不同 OS、攝影狀態等會影響辨識的精確度，因此雖然在 Jetson Nano 端可以通過 GPU 與較大的 swap 有效地提升程式效能（最高約 27FPS），但其精確度不高。（除筆電外均通過 Pi Camera v2.1 取得影像資料）

	FPS	辨識精度	OS
Jetson Nano（有 GPU）	27	低	Jetson Nano 官方 OS
RPI3	2	中	Raspberry Pi OS 64bit
RPI4	Unknown	未知	未知
MacBook Air (M1 8GB)	19	高	macOS Monterey

手勢設定：
啟動狀態：✌(G): Go, ✋(S): Stop
移動方向：👈(L): Left, 👉(R): Right, ☝(F): Front, 👇(B): Back
速度控制：  (U): Speed Up,  (D): Slow Down

在 Mbed 端，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 OS Fault Error。

以下是功能四之專案中使用到的一些上課教過的內容(並無全部列出，僅列出一些代表性的):
在 SocketModule.cpp 中:

```
_sem->acquire();
// printf("received %d bytes:\r\n%.*s\r\n\r\n", received_bytes, strstr(buffer, "\n") - buffer, buffer);
// printf("received %d bytes:\r\n%.*s\r\n", received_bytes, result, buffer);
printf("received %d bytes:\r\n%s\r\n", received_bytes, buffer);
_sem->release();

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();
}
else if(strstr(buffer, "Back")){
    _mutex_motor->lock();
    *_motor_state = 'B';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "SpeedUp")){
    _mutex_motor->lock();
    *_motor_state = 'U';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
else if(strstr(buffer, "SlowDown")){
    _mutex_motor->lock();
    *_motor_state = 'D';
    _cond_motor->notify_all();
    _mutex_motor->unlock();
}
```

5. 功能五：遙控 BLE

```
void MyGattClient::modify_BLE_device(char* device_name){
        _ble->gap().disconnect(ble_module.get_connection_handle_t(),
        ble::local_disconnection_reason_t::USER_TERMINATION);
        ble_module.stop();
        ble_process.modify_peer_device_name(device_name);
}
```

通過此函式可以重置已連接的 BLE Instance，並將連線的 GATT Server 目標設置為對應的名稱，通過將 myGattClient 傳入 mySocket 使我們可以直接在 receive_http_response 直接呼叫對應函式，並同上通過 cond_socket 與 buffer 傳送對應的 characteristic 變化等。

以下是功能五之專案中使用到的一些上課教過的內容(並無全部列出，僅列出一些代表性的):

在 MyGattClient.cpp 中:

```
MyGattClient::MyGattClient(BLE &ble, events::EventQueue &event_queue, Semaphore &sem, Mutex &mutex, ConditionVariable &cond, char* buffer, bool &changed_ble, char* buffer_ble, bool &error_state, PinName led, char &mode):
  _ble(&ble), _event_queue(&event_queue), ble_process(event_queue, ble), _print_sem(&sem), _mutex(&mutex), _cond(&cond), _buffer(buffer), _changed_ble(&changed_ble), _buffer_ble(buffer_ble), _error_state(&error_state), led(led), _mode(&mode){
}
```

```
void MyGattClient::start(){
    _thread.start(callback(this, &MyGattClient::setup));
    _thread_led.start(callback(this, &MyGattClient::setup_led));
    // _thread_modify.start(callback(this, &MyGattClient::setup_modify));
}
```

```
void MyGattClient::setup(){
    ble_module.setup(*_print_sem, *_mutex, *_cond, _buffer, isConnected, *_mode);
    ble_process.on_init(mbed::callback(&ble_module, &GattClientModule::start));
    ble_process.on_connect(mbed::callback(&ble_module, &GattClientModule::start_discovery));
    ble_process.start();
}
```

6. 額外功能：DSP

我們利用對 STM32 原先錄好的音檔(這邊指 raw data, in hexadecimal form or in binary form)進行差分方程(difference equation)來達成濾波的目的。注意到，對於兩個 discrete-time signal x[n]和 y[n](其中 x[n]為 input 訊號，y[n]為 x[n]通過濾波器後所得到的 output 訊號)，若$X(e^{jw})$和$Y(e^{jw})$分別代表它們的傅立葉轉換，則若$X(e^{jw})$和$Y\left(e^{jw}\right)$滿足以下的關係式:

$$\frac{Y(e^{j\omega})}{X(e^{j\omega})} = \frac{\sum_{k=0}^{M} b_k e^{-jk\omega}}{\sum_{k=0}^{N} a_k e^{-jk\omega}}$$

則 x[n]和 y[n]在 time-domain 的關係就如同下式所刻劃:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

此即為一遞迴關係式(recursive equation)，從 programming 的角度來看相當好處理。換句話說，只要我們能 specify 好係數$b_0, b_1, \dots, b_M$以及$a_0, a_1, \dots, a_N$，使得$\frac{Y\left(e^{jw}\right)}{X\left(e^{jw}\right)}$在頻域中表示一個 lowpass filter，則上述的 recursive equation 所算出來的 output y[n]就是通過一個 lowpass filter 後所過濾出來的訊號了。對此，我們使用 MATLAB 去生成需要的係數$b_0, b_1, \dots, b_M$以及$a_0, a_1, \dots, a_N$，並且用 STM32 來 implement 上述的 recursive equation。

以上是 filter 理論方面的敘述，下面我們進入程式碼的講解。

```
static size_t TARGET_AUDIO_BUFFER_NB_SAMPLES = AUDIO_SAMPLING_FREQUENCY; //record 1 seconds
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));
```

注意到，我們設定 AUDIO_SAMPLING_FREQUENCY 為 16000。又由於只錄一秒，故 TARGET_AUDIO_BUFFER_NB_SAMPLES 也就設為 16000(只錄一秒的原因是因為 STM32 的記憶體不足了，這部份我們會在 DSP 的成果中探討)。

在此我們開兩個 buffer: TARGET_AUDIO_BUFFER 以及 TEMP_TARGET_AUDIO_BUFFER，它們都是裝有 int16_t 資料的 array，其中 TARGET_AUDIO_BUFFER 用來裝麥克風直接錄到的音檔，而 TEMP_TARGET_AUDIO_BUFFER 則是用來裝 filtered 過後的音檔。

```
//create a 3-order butterworth lowpass filter, 3db freq = 500hz
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;
```

```
//create a 3-order chebyshev type-1 lowpass filter, 3db freq = 516hz
static double b_0 = 0.2884*0.001;
static double b_1 = 0.8652*0.001;
static double b_2 = 0.8652*0.001;
static double b_3 = 0.2884*0.001;

static double a_0 = 1;
static double a_1 = -2.8274;
static double a_2 = 2.6949;
static double a_3 = -0.8652;
```

```
//create a 3-order elliptic lowpass filter, 3db freq = 516hz
static double b_0 = 0.3436*0.001;
static double b_1 = 0.8111*0.001;
static double b_2 = 0.8111*0.001;
static double b_3 = 0.3436*0.001;

static double a_0 = 1;
static double a_1 = -2.8274;
static double a_2 = 2.6950;
static double a_3 = -0.8652;
```

以上是不同濾波器所需要的係數$b_0, b_1, \ldots, b_M$以及$a_0, a_1, \ldots, a_N$，其中我們都選用三階濾波器(以方便比較不同濾波器所過濾出來的結果)，故 M=N=3。注意到，在實際程式的執行中，其他所有嘗試的濾波器都會註解掉，這邊只是為了方便報告的展示，故才會一次貼出三個不同濾波器的係數都沒有被註解掉的情況。

```
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,
    0x00, 0x7d, 0x00, 0x00, //  (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,
};
```

以上程式碼處理了 wav 檔中的 header 部分。

```
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];
        }
    }
```

以上是我們 implement recursive equation 的程式碼，其中 buf_16t 代表 input 訊號 x[n]，buff_temp 代表 output 訊號 y[n]。從這邊就可以看出 recursive equation 的好處，就是運算相對簡單，每次計算一個 output 訊號只需要 7 個加減運算即可。相較之下，若選用直接將 input 訊號 x[n]和某個脈衝響應 h[n]做 convolution 運算來實作濾波器的話，計算一個 output 訊號可能就要幾十個或者幾百個加減運算了(基本上取決於 h[n]非零的長度是多少)，這也是我們選擇用 recursive equation 來實作濾波器的原因。

```
uint8_t *buf8t = (uint8_t*)TEMP_TARGET_AUDIO_BUFFER;
for (size_t ix = 0; ix < TARGET_AUDIO_BUFFER_NB_SAMPLES * 2; ix++) {
    printf("%02x", buf8t[ix]);
}
```

最後，由於 wav 檔的格式需求，我們將 TEMP_TARGET_AUDIO_BUFFER 看作是一個包含 uint_8t 資料型別之資料的 array，並且依序將這些資料 print 出。Print 出來的資料可以再透過轉檔程序，轉換成人耳能夠聽的 wav 檔。

注意到，上述 DSP 的部分因為 STM32 記憶體的不足而無法和我們專案中的其他功能做整合，故在以上的作法之敘述中，我們並沒有將 STM32 收到的音訊傳輸到 server 端，也就沒有使用到 audio 專案中有使用到的 mutex 以及 condition variables。

以下是 DSP 專案中使用到的一些上課教過的內容:

```
12    static EventQueue eventqueue;
```

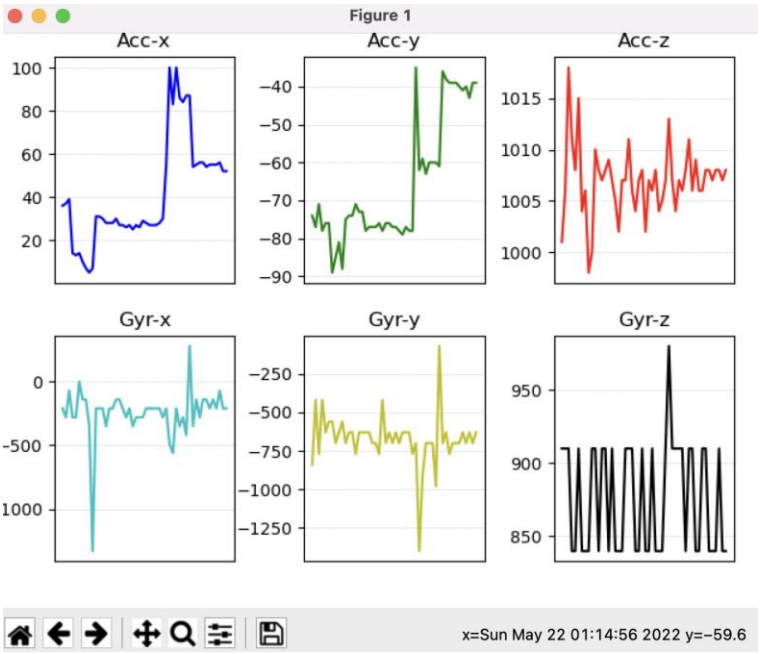
```
313     if (TARGET_AUDIO_BUFFER_IX >= TARGET_AUDIO_BUFFER_NB_SAMPLES) {
314         eventqueue.call(&target_audio_buffer_full);
315         return;
316     }
317 }
```

```
416     static InterruptIn btn(BUTTON1);
417     btn.fall(eventqueue.event(&start_recording));
418
419     eventqueue.dispatch_forever();
```

成果：

本次專案我們實現五大功能，分別為：資料圖像化、簡易雷達、音訊傳遞、手勢辨識遙控、遙控 BLE。此外，也包含未能整合進前面五大功能的 Audio-DSP。

a. 功能一：資料圖像化



（成功將 STM32 傳到 Server 端的加速度、陀螺儀位置的資訊給圖像化出來）

b. 功能二：簡易雷達

（車子前端的雷達探測器將探測之結果回傳後，我們將其視覺化出來）

c. 功能三：音訊傳遞

(1) 由於記憶體不足→downsample→傳到 server 端的音檔聽起來會有快進的現象

(2) 由於有無線傳輸之需要→對 STM32 錄好的音訊檔進行 Partition 後再傳輸→傳輸過程中會有 loss 產生→傳到 server 端的音檔聽起來會有不連續的現象

由於音訊不方便直接在 Report 中展示結果，請助教直接至 demo video 中查看結果：

https://www.youtube.com/watch?app=desktop&v=UVQB6Vfc58c&feature=youtu.be

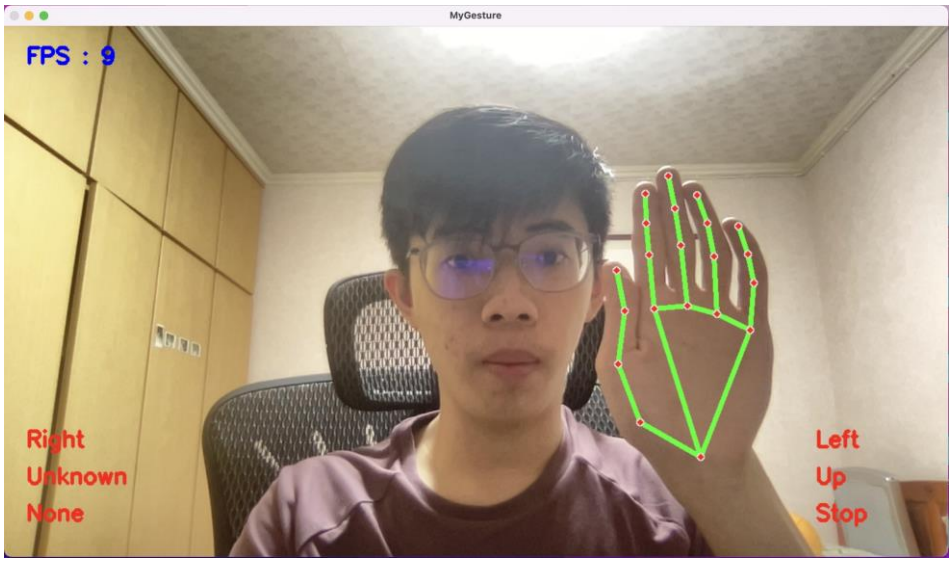
d. 功能四：手勢辨識遙控

手勢設定：

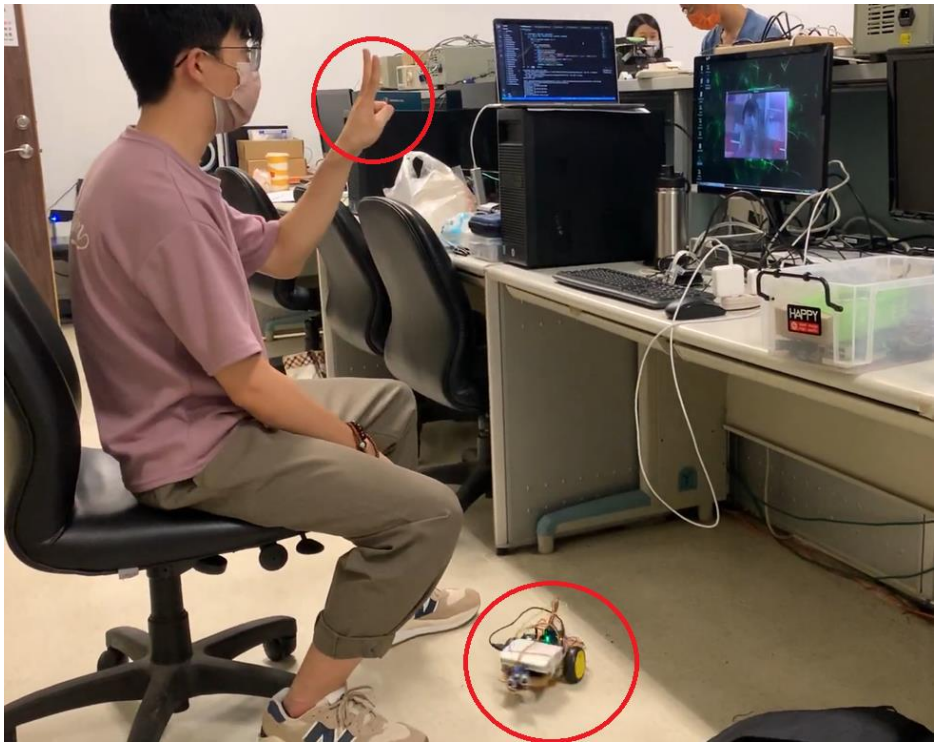
啟動狀態：✌(G): Go, ✋(S): Stop

移動方向：👈(L): Left, 👉(R): Right, ☝(F): Front, 👇(B): Back

速度控制：🤙(U): Speed Up, 👎(D): Slow Down



（上圖中可以看出手掌中的關節都有被正確地 identify 出來）

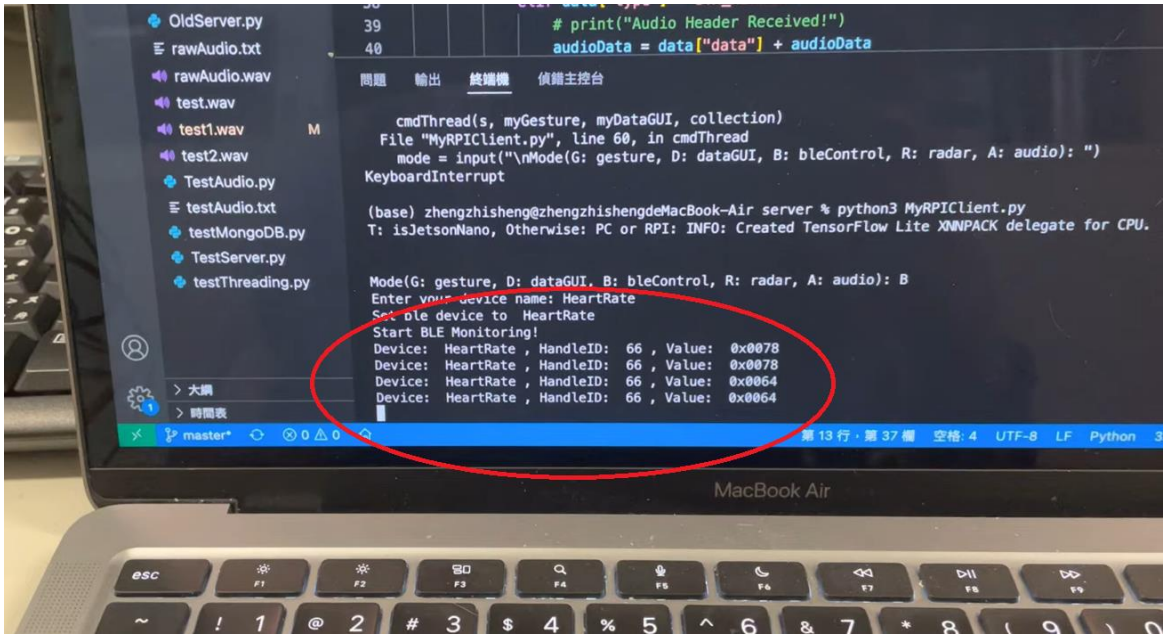


（上圖中，使用者比✌的手勢後，車子就開始旋轉了）

由於手勢控制的成果中有許多不同的手勢可以讓車子執行不同的動作，這部分 Report 中也不好展示，也請助教直接至 demo video 中查看成果：

https://www.youtube.com/watch?v=I0MvmhiXyvU

e. 功能五：遙控 BLE



（上圖為 STM32 收到藍芽裝置的 Charateristics, Services 之資訊後，傳送到 Server 端，在 Server 端所顯示之結果）

由於遙控 BLE 需要三方裝置的互相配合：STM32、藍芽裝置(在 demo video 中就是以 iPhone 為例)以及 Server 端，故也不太方便在 Report 中呈現出來，這部分也請助教直接至 demo video 中查看成果：

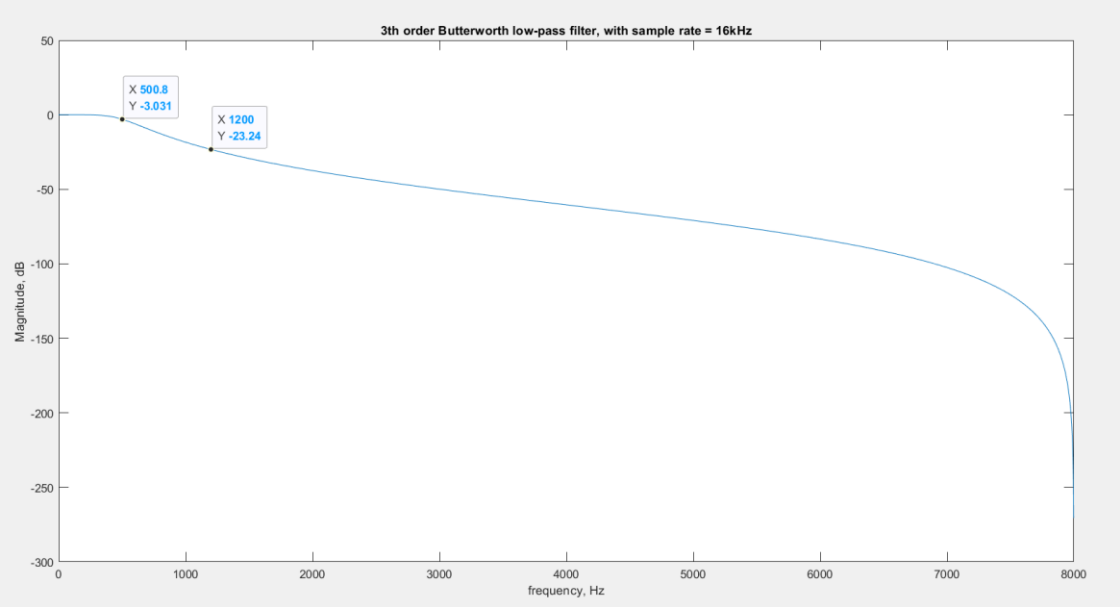
https://www.youtube.com/watch?v=5mQlsj9Lum0

f. 額外功能：DSP

我們展示三種不同的濾波器的定量描述(波德圖)，以及各個濾波器濾波後的結果(wav 檔)。對於下列每一個濾波器，我們都以我們的人聲(男生的聲音)加上一個 1200Hz 的單一高頻音之混合來做為我們的濾波器輸入。值得一提的是，每次之濾波測試都是現場直接用 STM32 錄音，故雖然都是同樣的人發出來的聲音搭配一個 1200Hz 的單一高頻音，但因為是不同時間錄的，故輸入訊號並不會完全相同(identical)，而僅能說就人耳的聽覺來說是相當接近。

(1) Butterworth filter

在此我們使用 MATLAB 去生成一個 3-order lowpass filter, 其 cutoff frequency = 500Hz(i.e., 實際上 Bode plot 中從最高增益下降 3dB 的那個頻率，大約會是 500Hz)。其 Magnitude 的 Bode plot 如下:

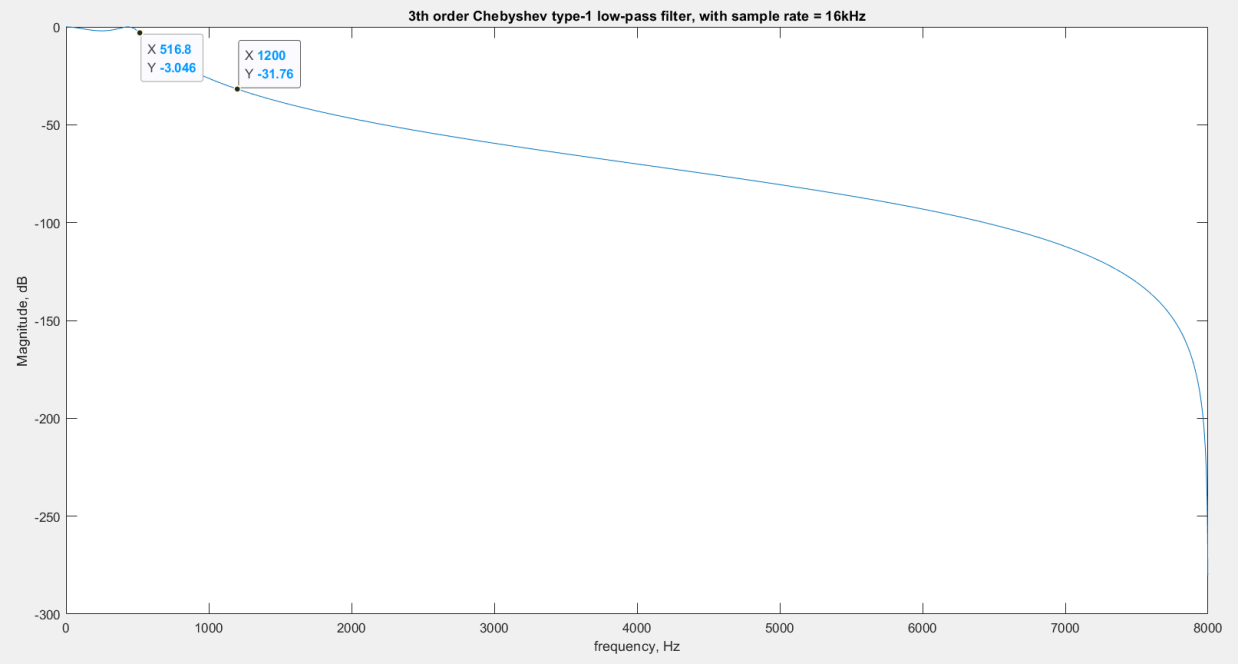


從上圖中可以看到，在頻率為 1200Hz 時，濾波器的增益大約為-23.24dB，這說明在過濾後的信號中，其頻域在 1200Hz 處的大小大約會是原始未過濾信號的 0.06887 倍。故我們可以合理預測，當輸入信號通過濾波器後，原先做為輸入信號一部分的 1200Hz 高頻音應該會合理衰減許多，而輸入信號中那些落在 passband 頻率中的人聲則應該不會有太大的衰減。此外，上圖中也可以看到 Butterworth 濾波器在 passband 的增益相當平緩，大約都為 0dB。

以下是我們的 wav 檔結果，其結果符合我們上述的預測。
https://drive.google.com/drive/folders/1oqx6rEp2oxpOlZCoA472Upo57LcWVrPb?usp=sharing
(由於 wav 檔無法上傳到 Youtube，故我們改上傳到 Google Drive，再請助教去雲端上下載了。)

(2) Chebyshev type-1 filter

在此我們使用 MATLAB 去生成一個 3-order lowpass filter, 其 cutoff frequency ≈ 516Hz(i.e., 實際上 Bode plot 中從最高增益下降 3dB 的那個頻率，大約會是 516Hz)。其 Magnitude 的 Bode plot 如下:

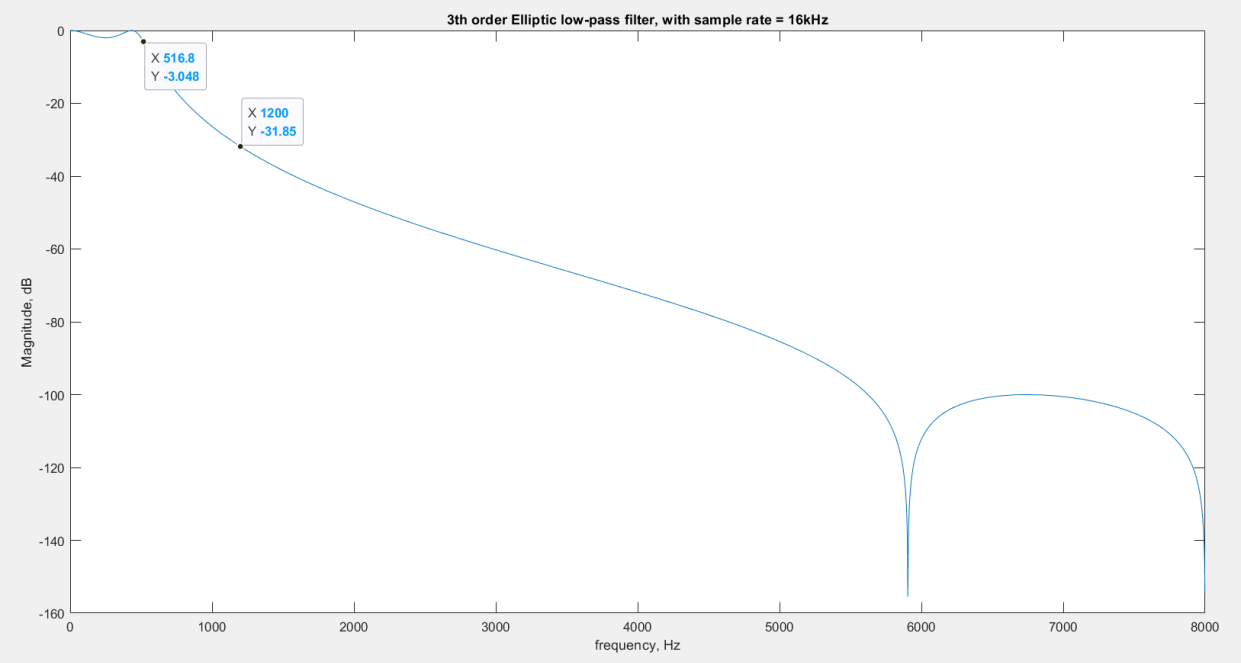


從上圖中可以看到，在頻率為 1200Hz 時，濾波器的增益大約為-31.76dB，這說明在過濾後的信號中，其頻域在 1200Hz 處的大小大約會是原始未過濾信號的 0.02582 倍。故我們可以合理預測，當輸入信號通過濾波器後，原先做為輸入信號一部分的 1200Hz 高頻音應該會合理衰減許多，而輸入信號中那些落在 passband 頻率中的人聲則應該不會有太大的衰減。此外，上圖中也可以看到，相較於 Butterworth 濾波器在 passband 的平緩增益，Chebyshev 型一濾波器在 passband 中有漣漪(ripple)存在，而漣漪的峰對峰值大約為 2dB，這將會導致輸入信號中那些落在 passband 頻率中的人聲在濾波後有些許失真(distortion)，因為人聲中不同頻率的聲音訊號將會有不相同的增益；但由於漣漪的峰對峰值大約為 2dB，故人聲中的不同頻率之聲波訊號，在濾波後，其彼此之間的大小差異最多不會超過原來之大小差異的$10^{0.1} \approx 1.259$倍，故就聽覺上來說，應該不會有明顯的失真。

以下是我們的 wav 檔結果，其結果符合我們上述的預測。
https://drive.google.com/drive/folders/13giiXzGY54alJThKd8DiIM0VUXop_f8q?usp=sharing
(由於 wav 檔無法上傳到 Youtube，故我們改上傳到 Google Drive，再請助教去雲端上下載了。)

(3) Elliptic filter

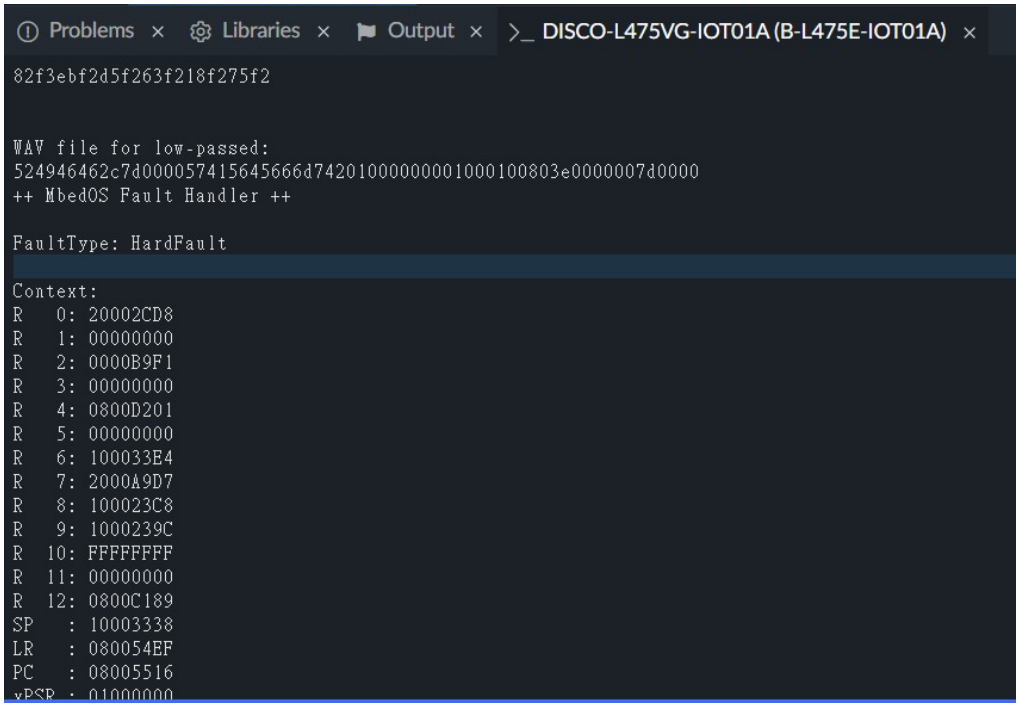
在此我們使用 MATLAB 去生成一個 3-order lowpass filter, 其 cutoff frequency ≈ 516Hz(i.e., 實際上 Bode plot 中從最高增益下降 3dB 的那個頻率，大約會是 516Hz)。其 Magnitude 的 Bode plot 如下:

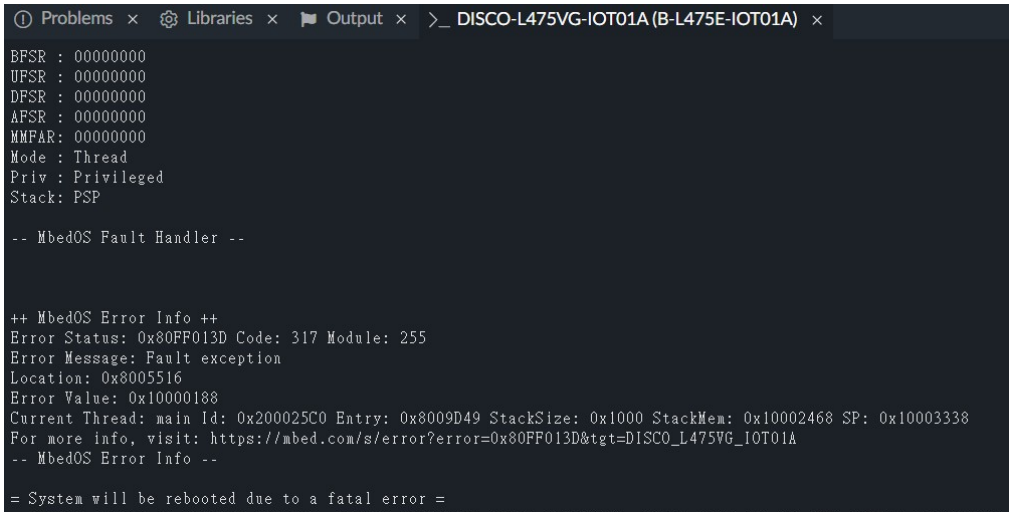


從上圖中可以看到，在頻率為 1200Hz 時，濾波器的增益大約為-31.85dB，這說明在過濾後的信號中，其頻域在 1200Hz 處的大小大約會是原始未過濾信號的 0.02556 倍。故我們可以合理預測，當輸入信號通過濾波器後，原先做為輸入信號一部分的 1200Hz 高頻音應該會合理衰減許多，而輸入信號中那些落在 passband 頻率中的人聲則應該不會有太大的衰減。此外，上圖中也可以看到，相較於 Butterworth 濾波器在 passband 的平緩增益，Elliptic 濾波器在 passband 中亦有漣漪(ripple)存在，而漣漪的峰對峰值大約為 2dB，這將會導致輸入信號中那些落在 passband 頻率中的人聲在濾波後有些許失真(distortion)，因為人聲中不同頻率的聲音訊號將會有不相同的增益；但由於漣漪的峰對峰值大約為 2dB，故人聲中的不同頻率之聲波訊號，在濾波後，其彼此之間的大小差異最多不會超過原來之大小差異的$10^{0.1} \approx 1.259$倍，故就聽覺上來說，應該不會有明顯的失真。

以下是我們的 wav 檔結果，其結果符合我們上述的預測。
https://drive.google.com/drive/folders/1ox5oZgV6B6zfe68i3frd2h_tMbqF4kuC?usp=sharing
(由於 wav 檔無法上傳到 Youtube，故我們改上傳到 Google Drive，再請助教去雲端上下載了。)

另外，誠如我們在 final presentation 所講的，DSP 的部分因為 STM32 的記憶體不足而無法與其他應用進行整合。根據我們的嘗試，當錄音的 sampling frequency 設為 16000Hz 且我們開兩個 buffer 來儲存音訊時(一個 buffer 用來儲存 unfiltered 的音訊檔，而另一個 buffer 用來儲存 filtered 過後的音訊檔；而音訊的每個 sample 為 16bit 的整數型別資料)，在僅錄製一秒時，這兩個 buffer 會需要 16000*2*2=64000bytes 的記憶體空間，此時可以順利錄製並進行 DSP 處理；然而，當我們要錄製兩秒時，理論上這兩個 buffer 會需要 16000*2*4=128000bytes 的記憶體空間，但此時 Mbed OS 已經跳 HardFault error 了，表示 memory 的部分有錯誤產生。





總結：

本次專案我們實作的專案沒有特別複雜的演算法，但龐大的專案架構與多執行緒的流程控制與資源分配是我們遇到的主要挑戰，通過各種 OS API 與有效的物件化開發可以有效提升偵錯、維護與擴充功能等的效率。在硬體端，硬體資源的有效分配也是很大的挑戰，從運算效能到記憶體等各種問題都給我們帶來不小的麻煩，但也在過程中學到很多相關知識。此外，在 Pyhton 端也使用了許多不同的模組來幫助我們提供較精美的圖像化畫面，但面對不同模組間的整合也是我們最後採取命令列模式的 Client 程式的原因。

報告統整：

1. Proposal 報告：https://github.com/Sam-0403/ESLAB_Final_Project/blob/master/1.FinalProjectProposal.pdf

2. 進度報告：

鄭至盛: https://github.com/Sam-0403/ESLAB_Final_Project/blob/master/2.FinalProjectReport1.pdf

李晨滔: https://drive.google.com/file/d/1WSve7lQPhMlQIDMYNt0hBAA1Uf_ODUOs/view?usp=sharing

3. 期末報告：https://github.com/Sam-0403/ESLAB_Final_Project/blob/master/3.FinalProjectFinalReport.pdf

參考文獻或資料：

Wav Header 相關資料：https://onestepcode.com/read-wav-header/

Wav 檔案編輯：https://blog.csdn.net/qq_44420246/article/details/104851995

Jetson Nano 相關設定：https://automaticaddison.com/how-to-set-up-the-nvidia-jetson-nano-developer-kit/

Mbed OS microphone example：https://github.com/janjongboom/b-l475e-iot01a-audio-mbed

MP34DT01：https://www.st.com/en/audio-ics/mp34dt01-m.html

雷達 GUI 設定：https://makersportal.com/blog/2020/3/26/arduino-raspberry-pi-radar

Butterworth filter by MATLAB command: https://www.mathworks.com/help/signal/ref/butter.html

Chebyshev type-1 filter by MATLAB command: https://www.mathworks.com/help/signal/ref/cheby1.html

Elliptic filter by MATLAB command: https://www.mathworks.com/help/signal/ref/ellip.html

信號與系統課本: Oppenheim & Willsky, "Signals and Systems", 2nd Edition, 1997, Prentice Hall