2023 Spring ESLab Final Project

Remote Control by Clapping

B08901113 吳紹睿

B08901112 林家弘

Github: https://github.com/WuShaoJui/111_2_ESLAB_Project

動機:

智慧家電為生活帶來了更多便利性，可以自動調節燈光、溫度以及播放音樂等等，已經成為未來科技的趨勢。現今市場上已有不少產品，例如利用手機App、物聯網或是聲音等等來控制各種家庭設備。而我們小時候大多都幻想過自己能用拍手來控制電燈開關，這樣睡覺時就不用下床自己關燈了。因此以小時候的夢想為基礎，我們決定製作出一個可以用拍手聲來控制周邊設備的智慧家電。

使用設備與材料:

STM32L475E、RPi 3、DHT11、LED * 2、七段顯示器 * 2、若干杜邦線。

架構:

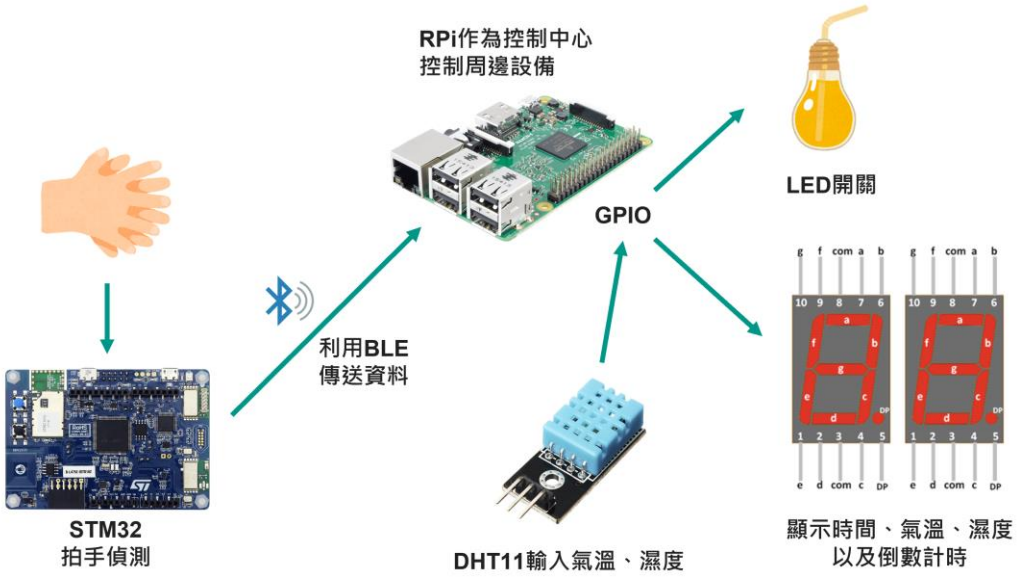


圖 1：架構圖

我們通過 STM32L475E 上的麥克風來記錄聲音，並在此板子上執行拍手偵

測的演算法。若 STM32 判斷為拍手聲，則將拍手聲的相對時間透過 BLE 傳送至 RPi。RPi 則作為周邊設備的控制中心，利用 DHT11 紀錄氣溫和濕度，並控制著 LED 和 2 個七段顯示器。

設計細節:

STM32 麥克風:

Sample rate: 16000 /s, Data rate: 32000 Byte/s

PCM buffer: 初步儲存麥克風資料，類似 cache。(size: 64 B)

Audio buufer: 儲存音訊資料。(size: 32000 B)

DMA 讓我們可以不用透過 CPU 將麥克風的 data 寫入 RAM 中，因此在錄音的過程中可以同時執行主程式。我們將麥克風的 data 先寫入 PCM buffer，且寫滿後會從頭寫入，避免占用過多記憶體。

Interrupt:

當 PCM buffer 半滿/全滿時，會觸發 half/complete transfer interrupt，此時可以在 ISR 中將 PCM buffer 的其中一半寫入後續處理資料用的 audio buffer。通過 sample rate 及 PCM buffer 大小計算可以得知，一次 transfer complete 會經過 $2\ \text{ms}\ (64\ \text{B} / (32000\ \text{B/s}) = 0.002\ \text{s})$，而後我們將透過 transfer complete 的次數來計算拍手當下的時間(相對於程式開始執行時)。

而為了進行拍手分析的演算法 audio buffer 大小設定為可以儲存 1 秒的音訊資料。當寫入 audio buffer 的 index 超過 buffer 長度時，將 index 歸 0 以達成循環寫入。

聲音數位訊號處理:

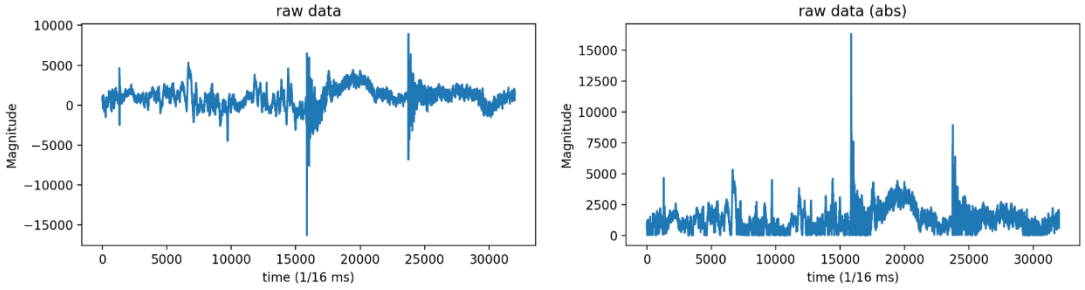


圖 2：錄音原始資料取絕對值比較

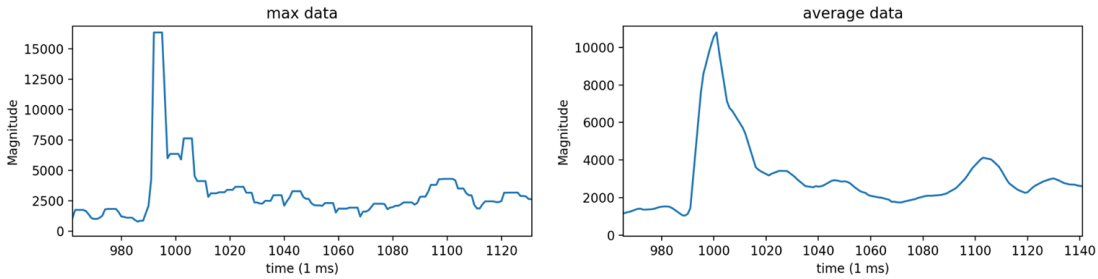


圖 3：每 16 筆資料取最大值(左)，再將資料取 5 筆移動平均(右)

我們以 audio buffer 的 index 所指的前一個位置的資料視為當下的聲音資料，而在接下來的演算法操作中，我們將以此位置作為原點，處理前一秒內的聲音資訊。

圖 2 為測試錄音(共 2 秒)，其中有兩聲拍手。我們先將聲音的數值取絕對值。而後(圖 3)，我們將絕對值資料每 16 筆取最大值，並且再將每 10 筆最大值做移動平均。透過這個做法，可以將看到曲線變得更加平滑，以便我們做後續的判斷。

拍手偵測演算法:

基本上是以處理後的資料有沒有超過 threshold 來偵測拍手，為了不誤判，要注意的是:

1. Threshold 需要根據背景雜音大小增減。
2. 由於拍手聲很短，超過 threshold 的時間不會太長。
3. 在室內拍手會有回音，需要設定最小間隔。

每 1 ms (一次 half transfer interrrupt)呼叫一次以下函數偵測拍手:

Algorithm clap_detect():

// 處理背景雜音

1. 算前 32 ms 的平均，將 threshold 加上此平均。

// 資料前處理，將數據平滑化

2. 取絕對值後，將每 16 筆資料取最大值，並每 10 筆最大值做移動平均。

// 處理拍手聲持續時間的限制

3. 如果處理過後的資料超過 threshold，將 exceed_threshold 設為 1，記錄超過 threshold 的持續時間，並 return。否則進入 step 4。

4. 如果處理過後的資料低於 threshold，且此時 exceed_threshold 為 1 (代表前面的資料有超過 threshold 一段時間)，進入 step 5，否則直接 return。
5. 若超過 threshold 的時間低於 32 ms，判斷為拍手，透過 BLE 傳送當下相對時間至 RPi。讓接下來 32 ms 的資料都跳過不處理 (避免回音影響)，最後將 exceed_threshold 設為 0 並 return。

BLE:

連接後設定 CCCD 為 0x0001 開啟 notify。偵測到拍手時，從 STM32 透過 BLE 傳送當下 transfer complete 的次數(int)至 RPi。而 RPi 收到 notification 後會將收到的資訊從 little endian 的 bytes 轉成 int。

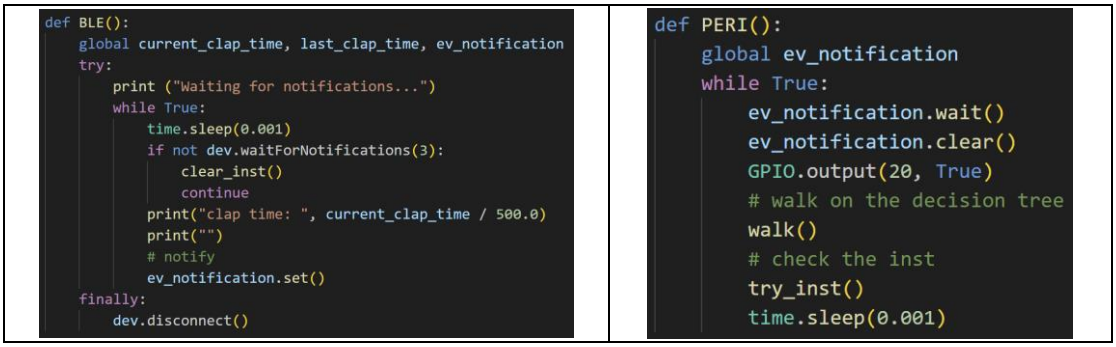
RPi Multithreading:

```
T_BLE = threading.Thread(target = BLE)
T_PERI = threading.Thread(target = PERI)
T_DHT = threading.Thread(target = update_DHT)
T_TIME = threading.Thread(target = display_time)
T_COUNT = threading.Thread(target = count_down)
```

圖 4：RPi 中所有的 thread

在 Central_RPi.py 中共有 5 個 thread：(在 RPi 連接上 STM32 後會開始執行)

1. BLE: 負責接收 notification，每 1 ms 偵測一次。
2. PERI: 收到拍手後(event)判斷指令並控制周邊設備。
3. update_DHT: 定期更新 DHT11 測量到的氣溫和濕度，每 1 s 更新一次。
4. display_time: 當顯示時間的指令執行時，每秒轉換顯示小時與分鐘。(因為顯示器只有 2 個，所以需要不斷切換)
5. count_down: 當倒數計時的指令執行時，需要不斷倒數，每 0.1 s 執行一次(低於 10 s 會顯示小數一位)。



```
def BLE():
    global current_clap_time, last_clap_time, ev_notification
    try:
        print ("Waiting for notifications...")
        while True:
            time.sleep(0.001)
            if not dev.waitForNotifications(3):
                clear_inst()
                continue
            print("clap time: ", current_clap_time / 500.0)
            print("")
            # notify
            ev_notification.set()
    finally:
        dev.disconnect()
```

```
def PERI():
    global ev_notification
    while True:
        ev_notification.wait()
        ev_notification.clear()
        GPIO.output(20, True)
        # walk on the decision tree
        walk()
        # check the inst
        try_inst()
        time.sleep(0.001)
```

圖 5：event for notification

其中(如圖 5)，有一個 event 代表收到 notification，PERI()會等待這個 event，再執行後續檢查指令。

RPi functionality:

我們固定指令長度為 4，並以拍手間隔來區分作為指令輸入的判斷依據。其中，短間隔(0.1 s ~ 0.7 s)代表 0，長間隔(> 0.7 s)代表 1。因此，我們總共有 16 個指令($2^4 = 16$)。

表 1：所有拍手可以控制的指令

Function	描述
0 ~ 9	設定七段顯示器至對應數字(倒數用)。
10	切換目前設定的顯示器至另一個顯示器。以七段顯示器右下小點代表目前設定中的顯示器。
11	倒數開始。左邊七段顯示器上的數字代表十位數，右邊代表個位數，根據顯示數字開始倒數(以秒為單位)。其中當數字低於 10 時會顯示至小數第一位。
12	LED 開關。
13	顯示濕度。
14	顯示溫度。
15	顯示時間，由於只有兩個七段顯示器，因此每秒會切換顯示時與分。顯示小時時右下小點會亮起，顯示分鐘時則否。

表 1 中的 Function 欄表示指令被 binary encoding 後所對應的數字，例如 function $6 = (0110)_2$ 為輸入 6 至指定七段顯示器，拍手輸入為 1 短間隔 + 2 長間隔 + 1 短間隔。

打錯指令或發現麥克風誤認雜訊時，需要將錯誤的指令清除，我們設定若 3 秒內沒有偵測到拍手，會清空目前輸入至一半的指令。圖 5 左側的 waitForNotification(3)和 clear_int()即在做此事。

成果:

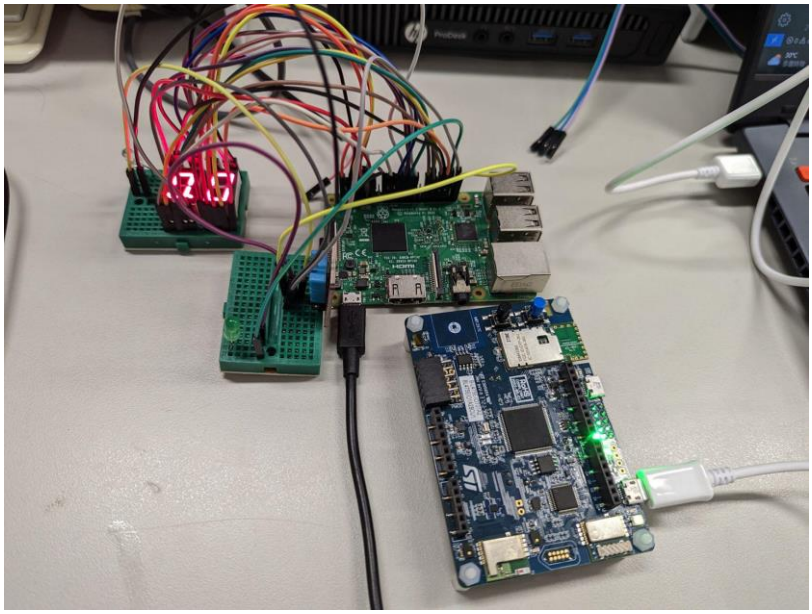


圖 6：最終實際電路

Demo 影片: https://youtu.be/Cxv2cGGojYg

分工:

- 吳紹睿：麥克風音訊處理、BLE、RPi Multithreading 與指令控制
- 林家弘：拍手偵測演算法、RPi GPIO 控制、接線師

參考資料:

[1] B-L475E-IOT01A-User-Manual-2019

[2] https://github.com/janjongboom/b-l475e-iot01a-audio-mbed

[3] https://pub.tik.ee.ethz.ch/students/2013-FS/GA-2013-03.pdf