Gesture Smart home Control

組員

B09901052 劉承亞、**B09901058** 邱奕翔

動機

現在大多數的家庭還是使用傳統的門鎖，所以當有人回家沒帶鑰匙，還是需要其他人跑到門口幫他開門，但很常會遇到無法馬上停下手邊事情的情況。又或者是已經躺在床上了，但是發現燈還沒關，此時又必須要下床關燈再回到床上。諸如此類的例子還有非常多，而這些目前需要我們親自去完成的事物其實有更方便的做法。因此，我們希望能透過一個穿戴裝置加上設定好的手勢，去完成在三樓幫家人開門，或是在床上關燈的操作。

介紹

我們的專案主要可分成四個模組：

Sensor (STM32)

Sensor的功能包含了偵測手勢，通過BLE偵測距離最近的Agent，並將這些資訊傳給Controller，去決定要對哪個Agent進行操作。

程式連結：https://github.com/EMbeddedNTU/final_sensor

Controller (RPi4)

Controller是中央控制的Server，負責接收Sensor的資料，發送控制Agent的Request。Agent也可以傳送通知給Controller，以通知前端。同時Contoller也負責和前端進行溝通，使前端能設置Sensor和Agent。

程式連結：https://github.com/EMbeddedNTU/final_controller

Frontend (IOS, Android, Web)

我們做了IOS, Android和Web的前端，我們希望使用者能透過友善的介面去設置Agent的資訊，控制Agent的狀態，以及各個手勢的功能。更可以透過前端監測Agent狀態，和接收通知。

程式連結：https://github.com/EMbeddedNTU/final_frontend

Agent (STM32)

Agent是我們系統中的實際應用裝置，在我們的Demo中，我們總共有兩個Agents，其中一個可以控制燈光，另一個可以控制燈光和電子門鎖（具有門鈴功能）。系統中可以有多個Agent，每個Agent可以同時具有多個功能。Agent還有狀態恢復的功能，在系統斷電之後能自行恢復先前的狀態。

程式連結：https://github.com/EMbeddedNTU/final_agent

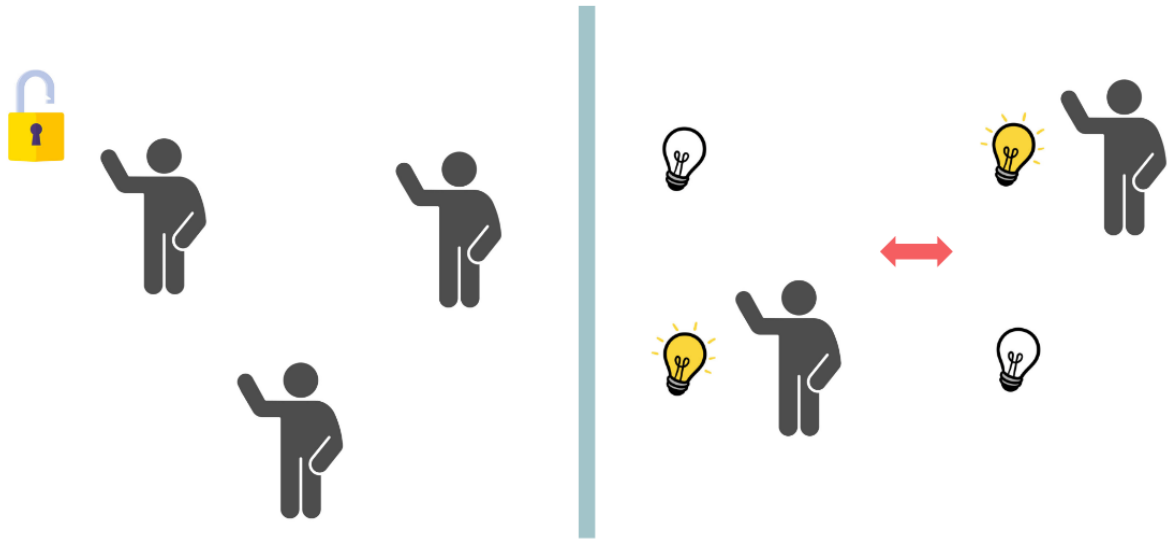
專案**GitHub**資料夾連結：**https://github.com/EMbeddedNTU/final_project**

專案版控設置

我們使用git submodule 的功能，將四個模組的程式放在不同的GitHub repository中，再透過submodule的方式加入一個完整的專案資料夾中。由於團隊中有兩人共同開發，而且不同模組間的版本可能會有相依性，使用submodule可以完美的解決此問題，每個模組可以分開開發，也可以確保在專案資料夾中四個模組的版本是可以共同運作的。

特色

手勢種類



左邊為全域手勢，代表在任何地方做此手勢都代表同一種意義，例子有開大門門鎖。右邊則是區域手勢，代表根據最近的Agent不同而有不同意義，例子有開一樓的燈以及開二樓的燈。

Mbed程式

我們sensor和agent都是使用STM32，會用到許多共通的功能，因此我們寫了一個core的函式庫，包含了許多我們會用到的基本功能，以下列舉：

FileSystem:

我們定義了一個FileSystem的class，在建立FileSystem instance後會先init()。這個class有開檔寫檔，印出檔案內容，建資料夾，進入資料夾等功能。

Logger:

我們定義了一個客製化的Logger，會以不同顏色呈現GSH_TRACE, GSH_DEBUG, GSH_INFO, GSH_WARN跟GSH_ERROR的訊息(GSH是GestureSmartHome的意思)，同時也會印出是在哪一行哪一個檔案被呼叫，讓除錯輕鬆不少。而且只要透過#define LOG_LEVEL就能控制哪些訊息要被印出，例如#define LOG_LEVEL_WARN就只會印出GSH_WARN和GSH_ERROR的訊息。

Socket Service (支援自動斷線重連) & Http Service:

我們定義了一個Socket的class，來封裝NetworkInterface, WifiInterface和TCPSocket的功能。其中包含wifi的掃描和連線，socket的連線、傳送和接收資料等等。另外我們的Socket可以支援自動的斷線重連，即使server斷電，只要重啟server後，client端可以自動連上。

```
bool init();

void wifi_scan();

bool wifi_connect(const char *ssid, const char *password,
                  nsapi_security security = NSAPI_SECURITY_WPA_WPA2);

bool wifi_connect_default();

bool connect(const char *hostname, const int port);

bool send(const char *buffer, int len);

int recv_chunk(char *buffer, uint32_t length);

void close();
```

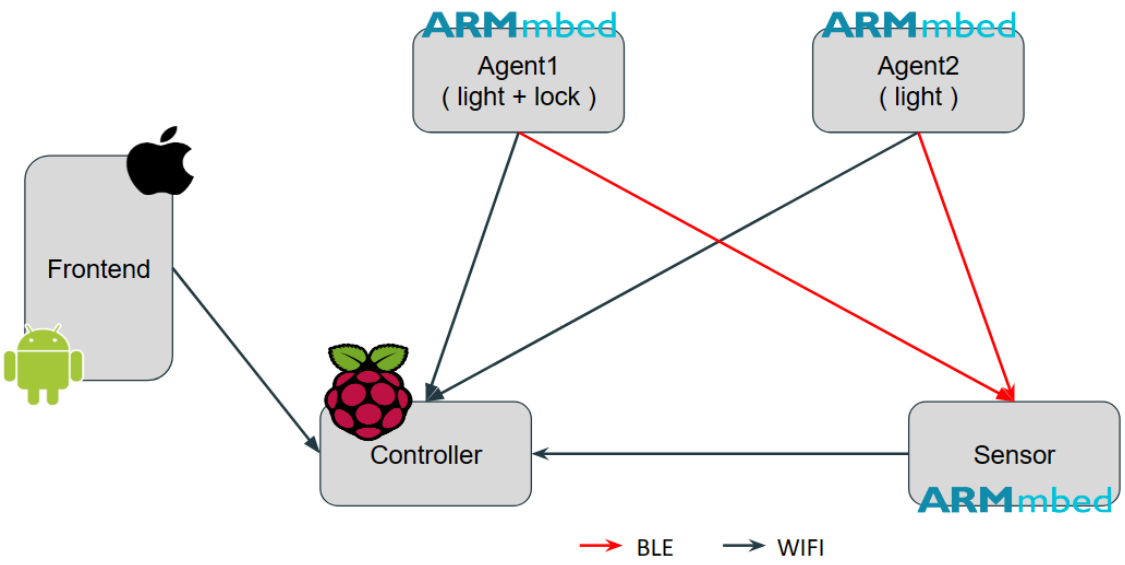
基於Socket之上，我們參考了外部的程式庫寫了一個Http service，Http service這個class以Singleton的方式實作，也就是說整支程式會有一個Http service的instance，只要透過static函數HttpService::GetInstance()就能取得。Http Service包含了get和post兩個方法，可以提供end point的url和額外的http header。

HttpResponse是我們定義的一個結構，包含了body, header, status code等等，我們使用了**SharedPtr**來防止memory leak。

```
SharedPtr<HttpResponse> http_get(const char *url, char *headers);
SharedPtr<HttpResponse> http_post(const char *url, char *headers, char *post_data);
```

作法

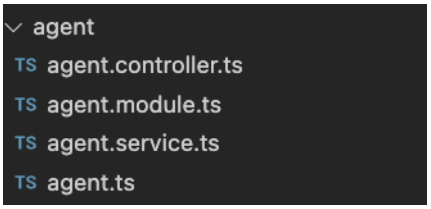
網路架構



上圖為我們的網路架構圖，Controller負責處理各種請求，為中央Server，Frontend會讀取Controller的設定檔，並顯示在畫面上供使用者修改。Agent則是會在最初向Controller註冊，並持續拿取最新狀態，也可以發送通知給Controller顯示於Frontend上。Sensor則透過BLE的RSSI值判斷自己距離哪個Agent最近，並在偵測到手勢同時將距離哪個Agent最近這項資訊一併傳給Controller，再由Controller透過設定檔去改變指定Agent的狀態。

Controller (RPi4)

我們的Controller使用了Nestjs，包含了多個Module，Agent module, Config module, Gesture module, Phone module, State command module和Notificaton module。

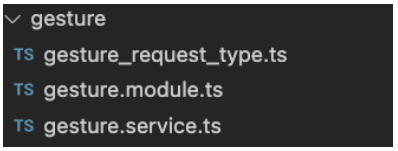


Agent module包含agent service和agent controller 其中agent controller 包含了三個API

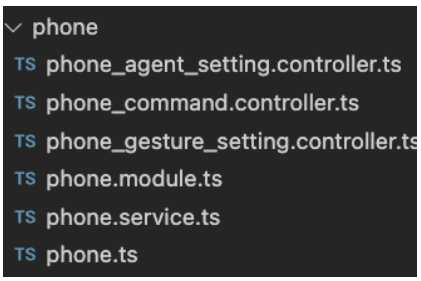
- register agent — 讓Agent註冊它的名稱、地點、功能及初始狀態，並存進config檔
- make gesture — 讓Sensor傳送它偵測到的手勢以及最近的Agent，並讀取config檔上面的資訊並改變相應Agent的State，再存回config檔
- get state — 讓Agent得知它目前的狀態

agent.ts中定義了Agent相關的classes。

Config module負責儲存和讀取Agent和Gesture的設定，將class instances轉成json的格式或將json轉成class instance。



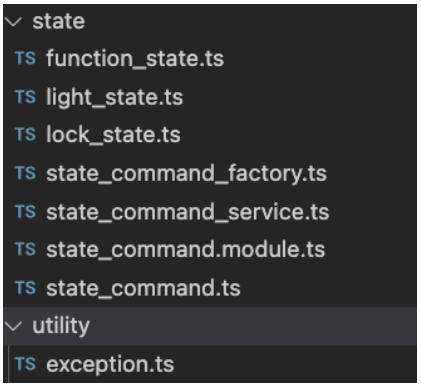
Gesture module中的gesture service透過前端傳回的資料，來新增和刪除config中的手勢設定。



Phone module放了前端多數的API，phone_agent_setting.controller.ts中包含了get agent profile和edit agent profile來取得或更新agent的資訊。

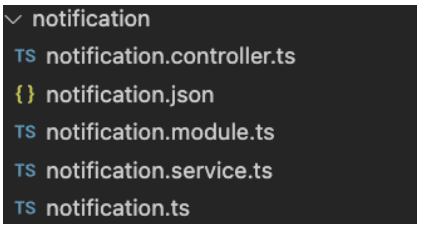
phone_command.controller.ts包含了make command這支API，讓前端能對Agent下指令。phone_gesture_setting.controller.ts 包含了六個API

- getGestureList — 拿到目前設定好的所有手勢
- getGestureOption — 拿到設定手勢時的可選選項
- getAllStateCommandOption — 拿到所有StateCommand
- getStateCommandOption — 根據AgentId拿到該Agent可選的所有StateCommand
- addGesture — 新增手勢
- deleteGesture — 刪除手勢



FunctionState用來儲存某個功能需記錄的所有變數，由於每個功能所需的變數都不一樣，因此FunctionState是一個base class，只會包含一個type。而light_state和lock_state會去implement FunctionState，分別包含燈光和鎖的狀態。而Command除了有Id和名稱外，他還存了一個function，這個function的tpye會是 (fState: FunctionState) => FunctionState也就是它會將一個state更新，這個state可以是任何implement

FunctionState的state。這樣的設計讓新增有新功能的Agent變得非常簡單，只要定義一個NewState 去implement FunctionState，以及一些Command就可以了。新增Command可以透過state_command_factory來完成，在state_command_service中有一個initStateCommand的函數，它會在controller server啟動時建立所有的command，讓這些command可以被使用。當某個手勢被做出時，我們會先透過目標的agent id找到agent的state，透過要執行的command id找到command (state_command_service中的getStateById函數)，然後將state丟進state command執行，再將回傳的state存回agent。



Notification module包含了notification service負責存取notification。以及notification.controller讓agent能透過post notification api來新增通知，前端透過get notification api取得通知。

Frontend (IOS, Android, Web)

前端的部分使用flutter進行開發，使用的語言為dart（語法類似C++），我們設計的整體架構包含components, const, environments, data, screens。

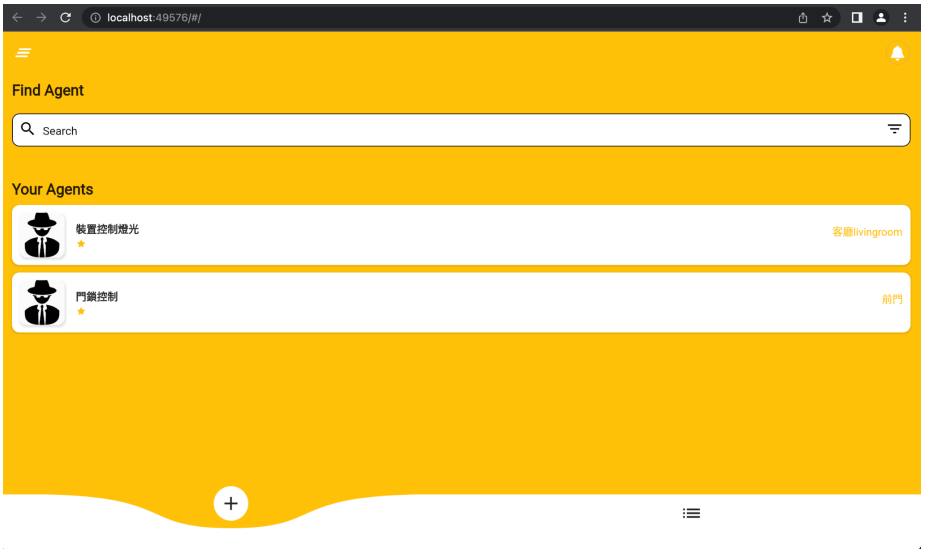
- **components** 包含了畫面中會使用的一些元件。
- **const** 包含App中使用的常數，包含了動畫和圖片等Assets的路徑。

- **environments**包含了針對不同環境的設置，由於我們平常會將後端架在localhost進行測試，但實際要用RPi作為後端，因此寫了兩個environment的config (local & develop)，讓我們可以快速的切換要build的版本。
- **data**負責資料流的處理，包含了所有資料型態的Data class, 將取得的response字串映射成這些class，以便處理。同時data中也包含了多個service（agent_service, gesture_service, notification_service），他們作為http client，取得不同頁面所需的資料，處理response mapping和status code判斷等等。
- **screens**處理畫面的呈現，每個畫面由model和page組成。model會在load畫面時，利用上面提到的service取得資料。另外model作為Change notifier會在資料改變時通知page，讓page重新顯示。所有的邏輯處理也都位於model中，page單純負責model資料的顯示。

以下圖片上面為手機畫面，下方為瀏覽器畫面：

Agent列表頁

- 列出所有有向controller註冊過的Agents
- 可以通過上方的搜尋欄透過名稱搜尋Agent
- 右上角鈴鐺在有通知時會顯示紅點 點擊可進入通知頁
- 點擊列表中Agent可進入Agent資訊頁

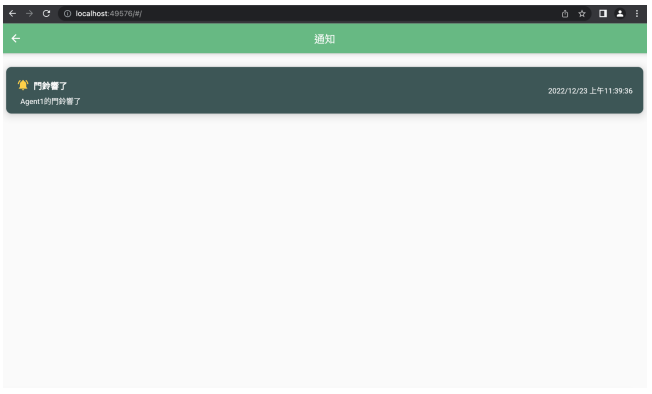


Agent資訊頁

- 顯示Agent名稱、位置(可以按編輯按鈕來更改)
- 再往下是此Agent能控制的項目及其狀態
- 以及能對此Agent執行的指令(最下方的紅色按鈕列表)

通知頁

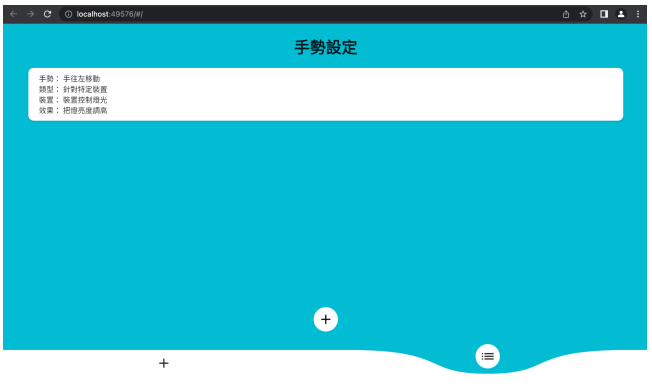
- **顯示通知**內容及時間



手勢頁

- 用來設置手勢的效果
- 新增時選擇手勢的動作、手勢的種類、作用的Agent，以及指令

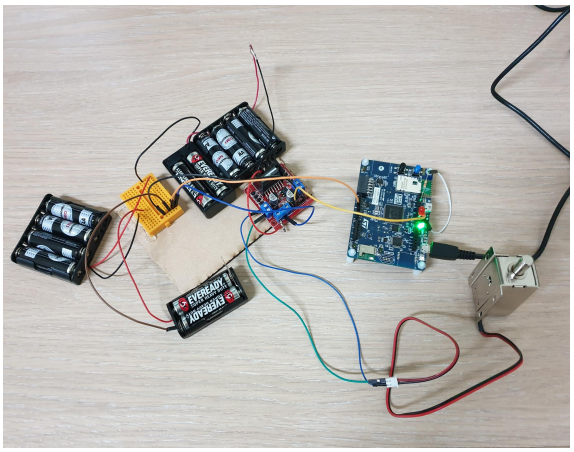
- 透過左滑或右滑刪除

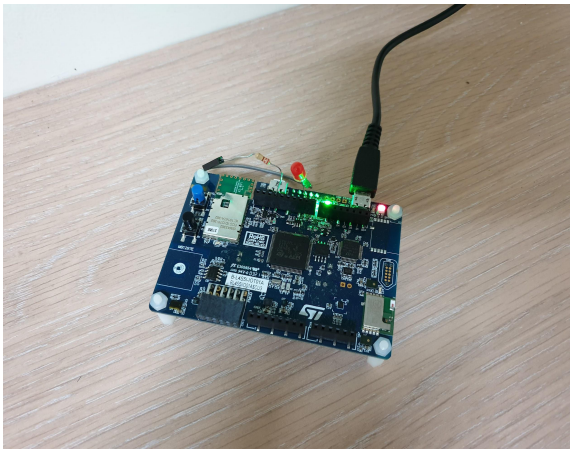


完整功能請參考影片(附在最下方)

Agent (STM32)

Type





左邊為Agent1，包含LED燈的控制以及門鎖的控制(利用L298N控制輸出電壓)。右邊則是Agent2，只包含LED燈的控制。

Register

Agent在剛開始的時候會向Controller註冊自己，包含提供自己的名稱、地點、支援的功能以及所有功能的初始狀態。

State

單一Agent可以具有很多功能，而每個功能都會有一個State，代表他的狀態。我們藉由Polling的方式持續獲得最新的State。而我們這次共有示範兩種State的形式。

LightState: Percent0, Percent25, Percent50, Percent75, Percent100

LockState: Locked, Unlocked

LightState是藉由PWM control來控制輸出電壓來調整亮度

LockState則是單純的輸出高低電位來控制開關鎖

State Recover

我們透過Mbed上面的FileSystem來實作還原狀態的功能，以達到在停電後所有Agent都會回復到原先狀態。我們的做法是透過每次讀取State時，就將更新的State存入Agent的state.txt當中。而Agent剛啟動時的第一步就是會去state.txt當中差看是否有上一次運行時記下來的state，若有則是用前一次的狀態作為Register時的初始狀態，沒有則是使用預設的狀態。

BLE Advertising

因為Sensor要知道距離哪個Agent最近，因此我們透過BLE的Advertising來實現。只要設定好Device name的開頭是Agent，就可以讓Sonser知道這個是其中一個Agent。

Notification

我們透過User button來模擬按門鈴的情況，只要具有門鎖功能的Agent上面的User button被按下，就會告訴Controller發生的事件以及觸發位置，再更進一步顯示於前端上。

Sensor

Sensor部分包含兩個部分，跑在各自的Thread上面，分別偵測手勢以及最近的Agent。

Gesture Detection

手勢辨識的部分我們實作了簡單的6個方位，分別是上下左右前後。而我們透過Threshold將值分為三個區塊，分別是大於正的Threshold、中間值以及小於負的Threshold。再根據簡化後的數據偵測是否出現特定的模式，以判斷是否有做相應操作。而不同方位的Threshold也不同，像是前後由於比較難做動作，因此Threshold比較小。而針對誤判我們有做以下四種排除。

- 對最近的10組數據做平均，避免出現極端異常值。
- 只在z軸加速度最大時進行偵測，排除平時手放下時的自然擺動。
- 動作時長必須足夠，避免手部的局部快速抖動造成誤判。
- 同時偵測到多個動作時將視為誤判，不採取任何動作。

```
[INFO: ./source/gesture.h:61]LEFT
[INFO: ./source/gesture.h:72]RIGHT
[INFO: ./source/gesture.h:61]BACK
[INFO: ./source/gesture.h:72]FRONT
[INFO: ./source/gesture.h:72]UP
[INFO: ./source/gesture.h:61]DOWN
```

BLE Scanning

在所有的BLE Advertising當中只針對Device Name開頭為Agent的裝置進行RSSI值的比較。而RSSI相對不穩定，所以我們嘗試了取平均以及取最大值兩種方法，但只要RSSI值出現偏差值，都與正常值差距極大，因此平均相較最大值更不穩定，故最終我們選擇了記住最大的RSSI值以及對應的裝置，並每5秒重置一次以確保獲得的資訊足夠及時。

成果

Frontend

- 前端畫面展示
- 搜尋**Agent** & 修改**Agent**資料
- 通知提示及頁面
- 對**Agent**執行指令
- 手勢設定

Agent

- 斷電後恢復斷電前狀態

Sensor

- 避免手部平時自然擺動以及局部快速抖動

Overall

- 手勢設定
- 燈亮度的四段控制
- 區域手勢開關燈
- 全域手勢開關燈
- 全域手勢控制門鎖

投影片

- ☐ **Team project proposal slide**
- ☐ 進度發表
- ☐ 嵌入式系統**DEMO**

參考文獻或資料

L298N教學

BLE Scanning

https://os.mbed.com/users/ollie8/code/Logger/

https://github.com/langhai/http-client-c

https://github.com/sangvaleap/flutter-app-food-ordering