

國立臺灣大學



嵌入式系統
車牌辨識停車管理系統

組員：林育新、邱士倫

指導老師：王勝德教授

摘要

車牌辨識停車管理系統在現今生活當中已經隨處可見，本專題的研究目的即是自己開發一個有相同功能的系統。採用樹梅派作為管理系統的中央電腦進行 AI 車牌辨識；STM32 作為閘門端的邊緣裝置，搭配 OLED、RFID、伺服馬達等模組；和 ESP32CAM 作為攝影鏡頭。裝置間透過 WIFI 連線。最終雖然執行速度稍慢於市面上現有產品，但作品功能完整，並且有相當高之準確度和擴充性。

Github 網址：

https://github.com/allenCHIUtw/Embed_finale.git

目錄

摘要.....	1
目錄.....	2
一、前言.....	3
二、材料介紹.....	4
三、系統架構.....	7
3.1 入場系統架構.....	7
3.2 離場系統架構.....	8
四、技術介紹.....	9
4.1 ESP32 CAM.....	9
4.2 STM32 與 Raspberry PI 的通訊.....	10
4.3 樹梅派上的車牌辨識.....	12
4.3.1 Dataset.....	12
4.3.2 Yolo 模型.....	12
4.3.3 模型的轉檔與量化.....	14
4.3.4 YOLO 模型的推論.....	16
4.3.5 影像預處理.....	17
4.3.6 車牌 OCR 辨識.....	18
4.4 STM32.....	19
4.4.1 RFID 模組.....	19
4.4.2 OLED 顯示螢幕.....	21
4.4.3 伺服馬達控制閘門.....	22
4.4.4 EVENT QUEUE.....	23
五、結果討論.....	24
5.1 實驗結果.....	24
5.2 未來展望.....	25
六、參考資料.....	26

一、前言

本專題的目標是設計並實現一個嵌入式車牌辨識停車管理系統，旨在降低停車場的人力管理成本。該系統將利用嵌入式硬體平台結合先進的車牌辨識算法，實現車輛的自動進出場記錄、費用計算及支付等功能。

二、 材料介紹

本次專案利用到 Raspberry PI 作為中央處理器，利用 ESP32 CAM 擷取車輛進出場時的資訊，發送訊息給 STM32 進行閘門的控制，用伺服馬達進行閘門的呈現。



圖 2.1、RaspBerry PI 3B，停車管理系統之中央電腦



圖 2.2、STM32 L475E，停車管理系統之閘門邊緣裝置



圖 2.3、ESP32 CAM & 8225N，攝影機與 Wi-Fi 模組



圖 2.4、伺服馬達，作為閘門制動的呈現

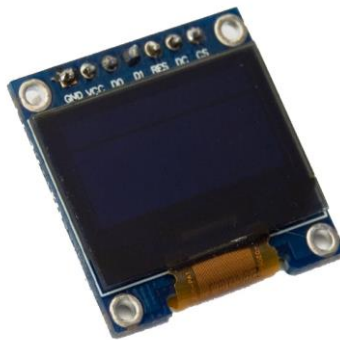


圖 2.5、SSD1306，OLED 在出入口顯示辨識結果

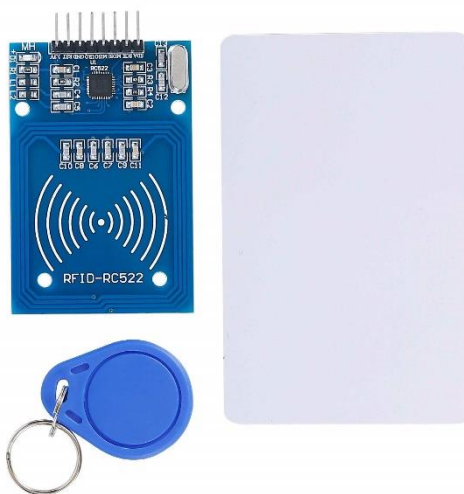
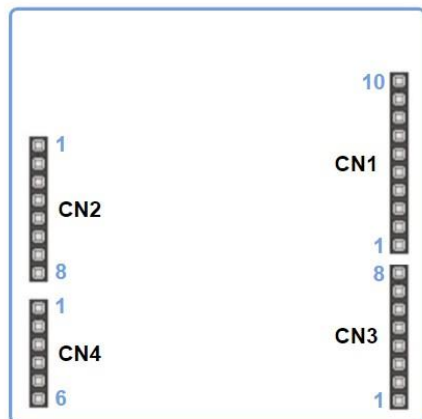


圖 2.6、RC522，RFID 感應模組



Connector	Pin number	Pin name	Signal name	STM32L 4+ pin	Function
CN2	1	ESV	-	-	-
	2	IOREF	-	-	3.3 V reference
	3	NRST	STM_Nrst	NRST	Reset
	4	3.3V	-	-	3.3 V I/O
	5	5V	-	-	5 V
	6	GND	-	-	GND
	7	GND	-	-	GND
	8	VIN	-	-	Power input
CN4	1	A0	ARD.A0-ADC	PC5	ADC
	2	A1	ARD.A1-ADC	PC4	ADC
	3	A2	ARD.A2-ADC	PC3	ADC
	4	A3	ARD.A3-ADC	PC2	ADC
	5	A4	ARD.A4-ADC	PC1	ADC / I2C3_SDA
	6	A5	ARD.A5-ADC	PC0	ADC / I2C3_SCL
CN1	10	SCL / D15	ARD.D15-I2C1_SCL	PB8	I2C1_SCL
	9	SDA / D14	ARD.D14-I2C1_SDA	PB9	I2C1_SDA
	8	AVDD	VDDA	-	VDDA
	7	GND	GND	-	Ground
	6	SCK / D13	ARD.D13-SPI1_SCK / LED1	PA5	SPI1_SCK / LED1
	5	MISO / D12	ARD.D12-SPI1_MISO	PA6	SPI1_MISO
	4	PWM / MOSI / D11	ARD.D11-SPI1_MISO / PWM	PA7	SPI1_MOSI / TIMxx
	3	PWM / CS / D10	ARD.D10-SPI1_SS / PWM	PA2	TIM2_CH3
	2	PWM / D9	ARD.D9-PWM	PA15	TIM2_CH1
	1	D8	ARD.D8	PB2	GPIO
CN3	8	D7	ARD.D7	PA4	GPIO
	7	PWM / D6	ARD.D6-PWM	PB1	TIM3_CH4
	6	PWM / D5	ARD.D5-PWM	PB4	TIM3_CH1
	5	D4	ARD.D4	PA3	TIMxx
	4	PWM / D3	ARD.D3-PWM / INT1_EXTI0	PB0	TIM3_CH3 / EXTI0
	3	D2	ARD.D2-INT0_EXTI14	PD14	EXTI14
	2	TX / D1	ARD.D1-UART4_TX	PA0	UART4_TX
	1	RX / D0	ARD.D0-UART4_RX	PA1	UART4_RX

圖 2.7、STM32 腳位設定

三、系統架構

3.1 入場系統架構

本研究在閘門裝置 (STM32) 上搭載 RFID (RC522) 的感應模組，當使用者靠近閘門以後刷卡，便會喚醒系統開始運作。首先閘門裝置會透過 WIFI-socket 通知中央電腦 (RPI) 去存取對應攝影機 (ESP32 CAM) 所拍攝到的相片內容。接著，中央電腦對相片使用深度學習與影像處理技術得到車牌資料，並將車牌資料與入場時間儲存至檔案中。下一步，回傳車牌資料給閘門裝置，由閘門裝置搭載的 OLED (SSD1306) 顯示辨識的結果，並且控制伺服馬達 (SG90) 開啟閘門。

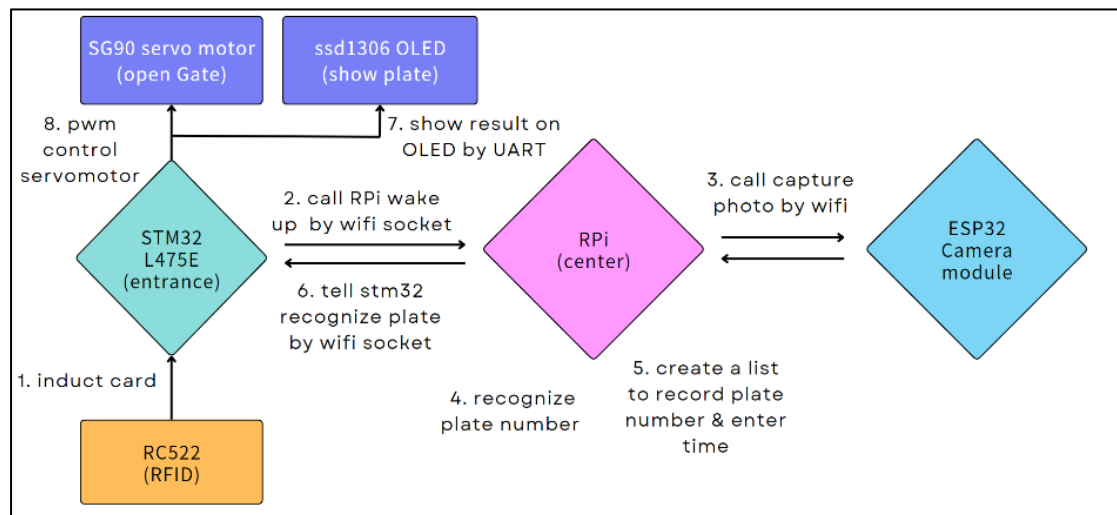


圖 3.1、車輛入場系統架構圖

3.2 離場系統架構

離場的系統架構與入場時十分相似。差別是，當辨識出車牌以後，需要查詢是否有對應的入場紀錄，並且最終將車牌、進出場時間一併回傳給閘門裝置當中顯示。

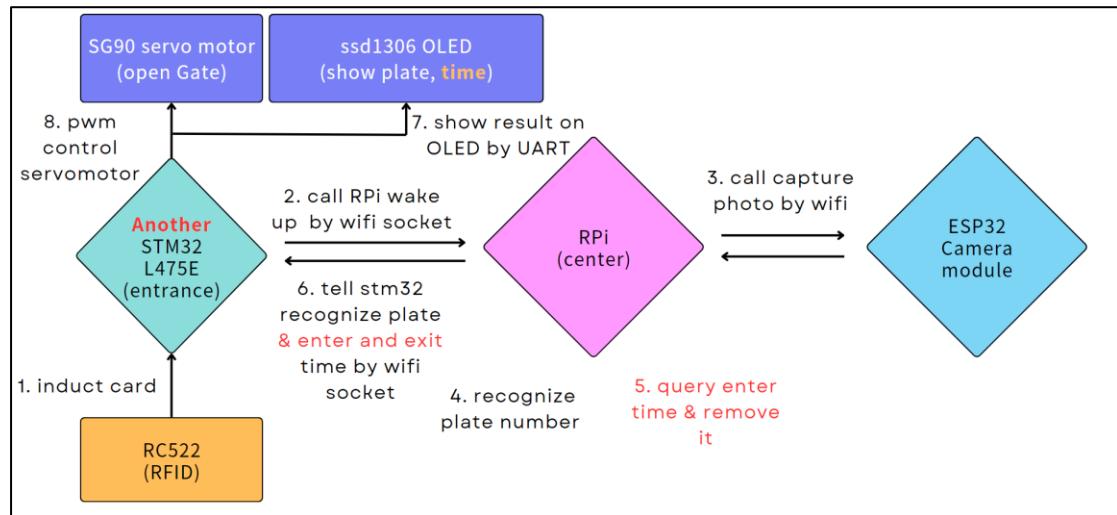



圖 3.2、車輛離場系統架構圖

四、技術介紹

4.1 ESP32 CAM

利用 WIFI 通訊，並預先設定好相機參數（圖片大小、品質、頻率等），透過 UART 輸出連接結果和獲得 IP 位置。



```
// =====  
// Enter your WiFi credentials  
// =====  
const char* ssid = "****";  
const char* password = "****";  
  
config.xclk_freq_hz = 5000000;  
config.frame_size = FRAMESIZE_UXGA;  
config.pixel_format = PIXFORMAT_JPEG; // for  
//config.pixel_format = PIXFORMAT_RGB565; //  
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;  
config.fb_location = CAMERA_FB_IN_PSRAM;  
config.jpeg_quality = 20;  
config.fb_count = 1;
```

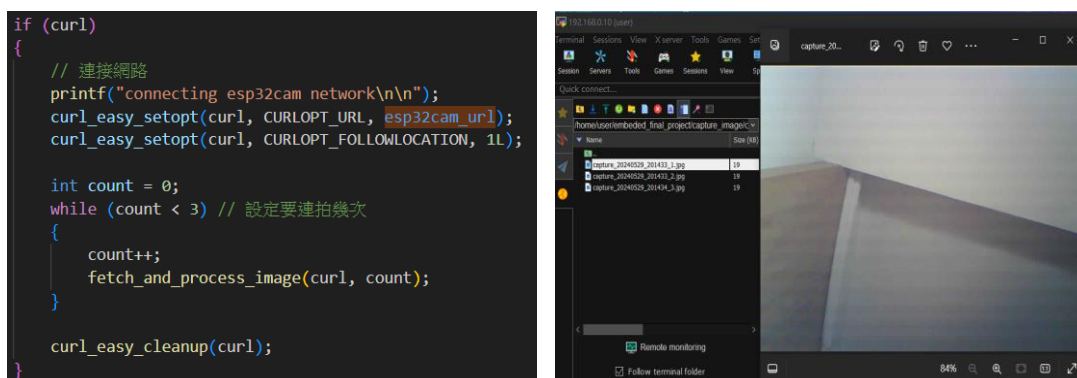
輸出 序列埠監控窗 x

訊息 (按 Enter 鍵將訊息發送到 COM11 上的 AI Thinker ESP32) 換行 115200 鮑率

WiFi connected
Camera Ready! Use 'http://192.168.0.20' to connect
...
WiFi connected
Camera Ready! Use 'http://192.168.0.20' to connect

圖 4.1.1、ESP32 利用 WIFI 連結及參數設定

在樹梅派當中透過 curl 存取網址，得到 ESP32 CAM 所拍攝的照片。本次專案中設定每次會擷取 3 張照片。



```
if (curl)  
{  
    // 連接網路  
    printf("connecting esp32cam network\n\n");  
    curl_easy_setopt(curl, CURLOPT_URL, esp32cam_url);  
    curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);  
  
    int count = 0;  
    while (count < 3) // 設定要連拍幾次  
    {  
        count++;  
        fetch_and_process_image(curl, count);  
    }  
  
    curl_easy_cleanup(curl);  
}
```

Terminal Sessions View X Window Tools Games Set
capture_20...
home/userembedded_final_projectcapture_image/c
W Name Size (KB)
capture_20240520_201431_1.jpg 19
capture_20240520_201431_2.jpg 19
capture_20240520_201431_3.jpg 19
Remote monitoring
Follow terminal folder
84%

圖 4.1.2、在 Raspberry PI 取得 ESP32 CAM 拍攝的照片

4.2 STM32 與 Raspberry PI 的通訊

本研究使用 WIFI-socket 作為 STM32 與 Raspberry PI 間通訊的方式，並且藉由傳送不同的訊息，告知 Raspberry PI 目前的閘門是設置為入口還是出口。當 STM32 送出訊息後，需要接收 Raspberry PI 回傳辨識完成的訊號以後，才能繼續執行顯示 OLED 和開啟閘門的動作。其中 Raspberry PI 回傳的訊息會包含" DONE" + " success" + {對應的車牌} + {進出場時間(出場時才有)}，或者" DONE" + " ERROR" + {錯誤訊息}。STM32 可藉由不同的回應，執行不同的操作。

```
bool send_http_request()
{
    const char buffer[] = "TRIGGER_ENTER";
    //const char buffer[] = "TRIGGER_EXIT";

    nsapi_size_t bytes_to_send = strlen(buffer);
    nsapi_size_or_error_t bytes_sent = 0;

    printf("\r\nSending message: \r\n%s", buffer);

    while (bytes_to_send) {
        bytes_sent = _socket.send(buffer + bytes_sent, bytes_to_send);
        if (bytes_sent < 0) {
            printf("Error! _socket.send() returned: %d\r\n", bytes_sent);
            return false;
        } else {
            printf("sent %d bytes\r\n", bytes_sent);
        }
        bytes_to_send -= bytes_sent;
    }

    printf("Complete message sent\r\n");

    return true;
}
```

STM32 發送

圖 4.2.1、STM32 端發送訊號

```

std::string receive_http_response()
{
    char buffer[MAX_MESSAGE_RECEIVED_LENGTH];
    int remaining_bytes = MAX_MESSAGE_RECEIVED_LENGTH;
    int received_bytes = 0;

    /* loop until there is nothing received or we've ran out of buffer space */
    nsapi_size_or_error_t result = remaining_bytes;
    while (result > 0 && remaining_bytes > 0) {
        result = _socket.recv(buffer + received_bytes, remaining_bytes);
        if (result < 0) {
            printf("Error! _socket.recv() returned: %d\r\n", result);
            return "Error run 01";
        }
        received_bytes += result;
        remaining_bytes -= result;
    }

    /* Null-terminate the buffer to make it a valid C-string */
    buffer[received_bytes] = '\0';

    /* the message is likely larger but we only want the HTTP response code */
    printf("received %d bytes:\r\n%.s\r\n\r\n", received_bytes, strstr(buffer, "\n") - buffer, buffer);

    /* Check if the buffer contains "DONE" */
    char* done_ptr = strstr(buffer, "DONE");
    if (done_ptr != NULL) {
        return std::string(done_ptr);
    }
}

```

STM32接收資訊

圖 4.2.2、STM32 接收訊號

```

if data == "TRIGGER_ENTER":
    print("接收到觸發訊號，開始執行命令")

    # 執行capture_image程式
    print("開始執行 capture_image")
    subprocess.run(["/home/user/embedded_final_project/capture_image/capture_image"], check=True)
    print("完成 capture_image")

    # 執行yolo的inference.sh腳本
    print("開始執行 yolo")
    subprocess.run(["/home/user/embedded_final_project/yolo/inference.sh"], check=True)
    subprocess.run(["python", "/home/user/embedded_final_project/yolo/test.py"])
    print("完成 yolo")

    # 執行recognize.py腳本，並取得結果
    print("開始執行 recognize.py")
    subprocess.run(["python", "/home/user/embedded_final_project/recognize/recognize.py"], check=True)
    result = subprocess.run(["python", "/home/user/embedded_final_project/recognize/enter_recognize.py"], capture_output=True, text=True, check=True)
    print(f"recognize.py 結果: {result.stdout}")
    print("完成 recognize.py")

    # 回傳執行完成訊號
    client_socket.sendall("DONE ".encode())

    # 回傳recognize.py的結果
    client_socket.sendall(result.stdout.encode())
    print("DONE \n")

```

圖 4.2.3、Raspberry PI 收到訊號後執行對應的程式

4.3 樹梅派上的車牌辨識

將樹梅派辨識車牌的任務，拆成 3 個子任務進行處理。因拍攝相片的部份，在 4.1 節 ESP32 CAM 時已經描述過，因此這個段落僅描述使用 yolo 對車牌進行初步裁切、預處理及車牌辨識的部份。

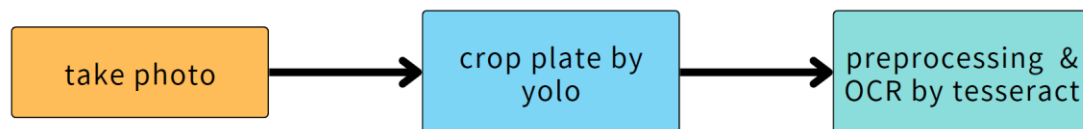


圖 4.3、車牌辨識子任務

4.3.1 Dataset

訓練資料庫為 Roboflow 上的 taiwan-license-plate-recognition-research-tlpr Image Dataset，此資料庫提供了約 20000 張的車牌資料。

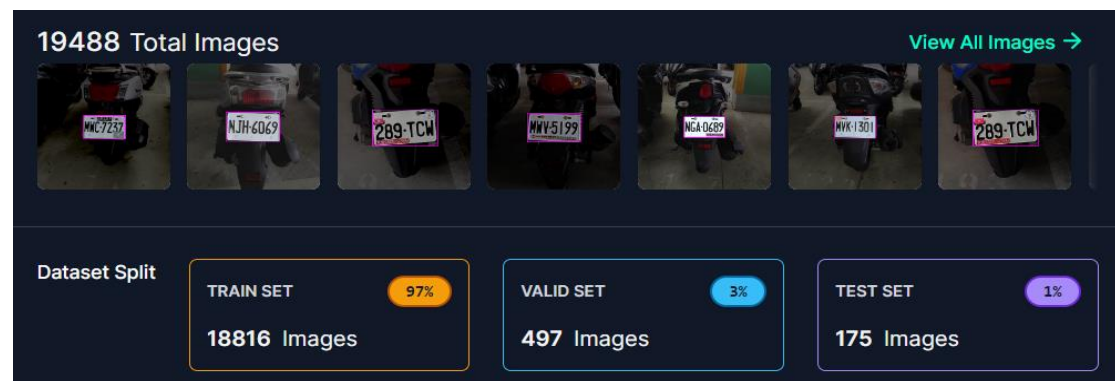


圖 4.3.1、選用資料庫

4.3.2 Yolo 模型

因為樹梅派的硬體限制，本研究採用 yolo v8 當中最小的 YOLO v8n 模型，推論單張圖片仍然需要約 2 秒的時間。但相較於 v8s 推論

單張圖片需要約 4 秒，已經有兩倍的效能提升。雖然是使用最簡單的模型，但因為是較簡易的偵測任務，並且資料量夠大，因此最後訓練的結果依然高達九成以上。

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

圖 4.3.2、不同版本 Yolo 所推論的單張圖片運算時間

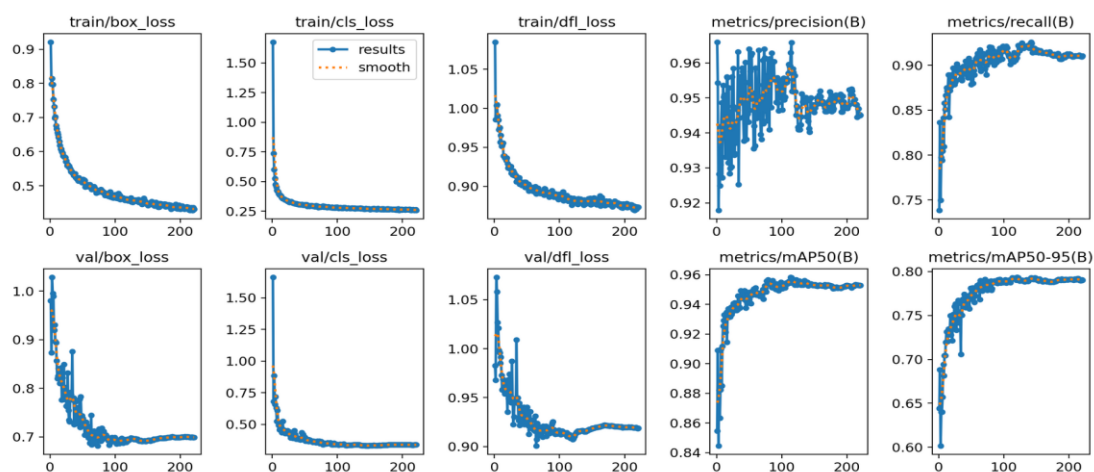


圖 4.3.3、訓練過程之損失與指標

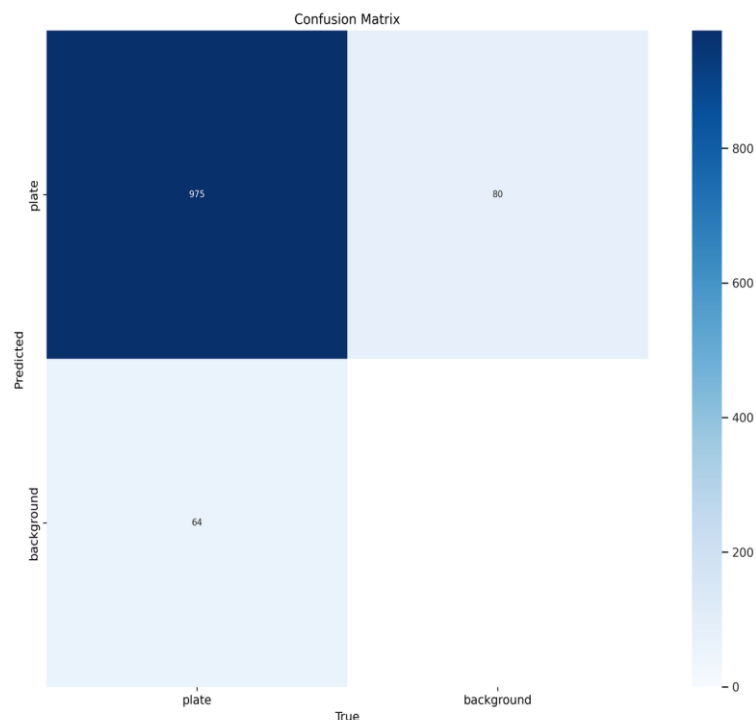


圖 4.3.4、訓練結果之混淆矩陣

4.3.3 模型的轉檔與量化

在推論以前，還需要將 yolo 產生的 .pt 模型檔案，轉換至 .onnx 檔案。ONNX 是一種開放式的模型格式。使得不同的人工智慧框架可以採用相同格式存儲模型數據並交互。在本研究實際推論的過程當中發現，在樹梅派 3B 當中執行推論任務時，直接使用 yolo 的檔案，每次約需耗費 30 秒左右，等待 ultralytics 的初始化，才能真正開始偵測車牌，造成效能嚴重瓶頸。

```
(yolov8) user@xrpri:~/embeded_final_project/yolo $ bash inference.sh
Ultralytics YOLOv8.2.19 Python-3.11.2 torch-2.3.0 CPU (Cortex-A53)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

WARNING ⚠ imgsz=[800, 608] must be multiple of max stride 32, updating to [800, 608]
image 1/11 /home/user/embeded_final_project/yolo/test_recognition/images/003257_jpg.rf.852351525c9712ec321f6a402bf6.jpg: 608x608 3 plates, 1945.9ms
image 2/11 /home/user/embeded_final_project/yolo/test_recognition/images/003267_jpg.rf.f207558cea7e044808e7bd89aa9d.jpg: 608x608 1 plate, 1849.1ms
image 3/11 /home/user/embeded_final_project/yolo/test_recognition/images/003279_jpg.rf.4f57b586ea48bae7d645f0ba2d8a.jpg: 608x608 1 plate, 1884.6ms
```

以上要等待約30秒

圖 4.3.5、直接推論每次都需多等待 30 秒

在搜尋資料後發現，如果將 yolo v8 訓練好的模型由 .pt 檔轉換為 .onnx 檔。推論時，便可以直接開始偵測車牌，對效能有極大改善。此外，還可以進一步量化模型。量化是指將模型中每個神經元的權重，從 float32 轉換為 int8 儲存。如此一來，檔案大小降低至 1/4，並且推論每張圖片速度從 2s 降低至 1.5s。

```
import torch
import onnx
from onnxsim import simplify

# 載入保存的.pth模型文件
model = torch.load("best.pt", map_location=torch.device('cpu'))
model = torch.load("skin-1600.pt", map_location=torch.device('cpu'))

# 將模型設置為推論模式
model.eval()

# 建立範例輸入
example_input = torch.randn(1, 3, 480, 480)

# 導出模型格式ONNX格式
onnx_path = "model.onnx"
torch.onnx.export(model, example_input, onnx_path, verbose=True)
print("ONNX 模型已保存:", onnx_path)
```

圖 4.3.6、將 pt 模型轉換為 onnx 格式

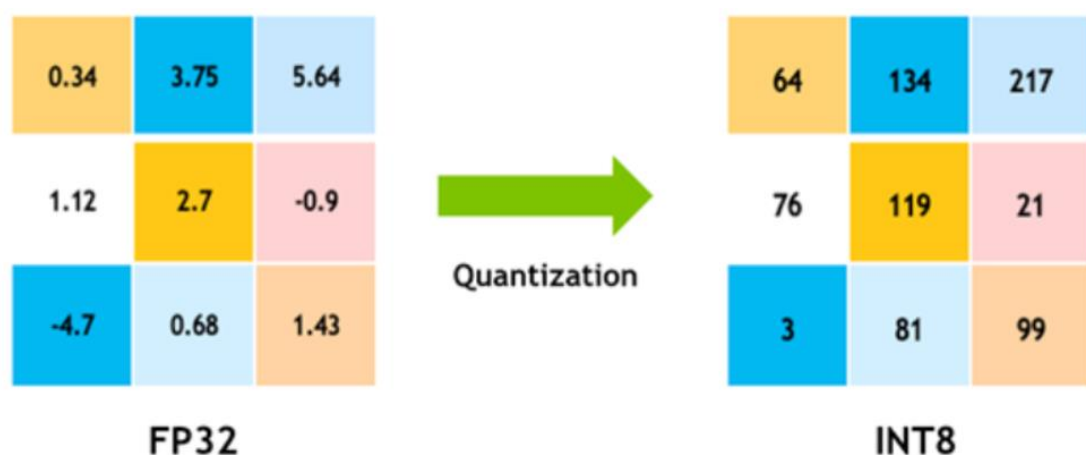


圖 4.3.7、量化模型

4.3.4 YOLO 模型的推論

在上一節當中提到，將模型轉檔與量化以後，可以提高推論的速度。但是與之相對的，推論的程式會變得相當複雜。對應的推論程式碼如下圖所示。

```
yolo predict task=detect model="/home/user/embedded_final_project/yolo/trained_weight/yolov8n/best.pt" source="/home/user/embedded_final_project/yolo/test_recognition/images/" save_crop=True imgsz=800,600
```

圖 4.3.8、推論 yolo 模型

```
64         boxes.append(box)
65         scores.append(maxScore)
66         class_ids.append(maxClassIndex)
67
68     result_boxes = cv2.dnn.NMSBoxes(boxes, scores, 0.25, 0.45, 0.5)
69     for index in result_boxes:
70         box = boxes[index]
71         box_out = [round(box[0]*scale), round(box[1]*scale),
72                   round((box[0] + box[2])*scale), round((box[1] + box[3])*scale)]
73         print("Rect:", names[class_ids[index]], scores[index], box_out)
74
75         # 保存裁剪後的圖像
76         x1, y1, x2, y2 = box_out
77         cropped_image = img[y1:y2, x1:x2] # 裁剪圖像
78         output_path = os.path.join(output_dir, f'{img_name}_cropped_{index}.jpg')
79         cv2.imwrite(output_path, cropped_image) # 保存裁剪後的圖像
80
81     # 記錄結束時間
82     end_time = time.monotonic()
83     processing_time = end_time - start_time
84
85     # 打印檔名和處理時間
86     print(f"Processed {img_name} in {processing_time:.2f} seconds\n")
87
88     print(f"裁剪後的圖像已保存到 {output_dir}")
```

圖 4.3.9、推論 onnx 模型

4.3.5 影像預處理

本研究利用 Open CV 對影像進行預處理，處理過程如下：1. 放大圖片尺寸、2. 進一步裁切車牌、3. 降噪。其中本研究也嘗試了增強對比度、灰階與二值化這幾種不同的預處理，但實測發現做了之後並沒有比較容易辨識出正確文字，因此棄用。而傾斜校正則是由於時間不足，因此沒有找到能夠穩定將圖片轉正的辦法。



圖 4.3.10、yolo 初步裁切後原圖



圖 4.3.11、放大尺寸、進一步裁切後



圖 4.3.12、降噪後

4.3.6 車牌 OCR 辨識

Tesseract 是一款著名的光學字元辨識引擎，本研究使用其對圖片做 OCR 車牌識別。進場時，若判斷辨識成功(字串長度=7)會將資料存進紀錄檔中。出場時，判斷辨識成功後，會至紀錄檔案中查詢資料，將進出場時間一併回傳給 STM32。

```
# 使用Tesseract進行文字識別
config = '--psm 6 -c tessedit_char_whitelist=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
text = pytesseract.image_to_string(binary, config=config)
print(f"plate text: {text}")
```

圖 4.3.13、使用 Tesseract 進行文字識別

```
(yolov8) user@yxrpi:~/embeded_final_project/recognize $ cat entry_log.txt
PTX8715,2024-06-05 21:26:23
```

圖 4.3.14、紀錄入場資料的檔案

4.4 STM32

4.4.1 RFID 模組

RC522 是一款 RFID 的模組，運作於 13.56MHz 頻率，並且透過 SPI 接口通訊。本研究從 mbed os 的官網找到已經實作好的函式庫：

https://os.mbed.com/teams/Project5_Software/code/RFID-RC522/docs/30e31f03d156/main_8cpp_source.html

但函式庫已經棄用許久，不支援新版的 mbed os 6，因此需要自行花費許多時間，將函式庫自行調整到新版本能用的樣子。最終修改過的函式庫，可見本專案儲存庫中的 Adafruit_OLED 資料夾。

```
if (!RfChip.PICC_IsNewCardPresent())
{
    snprintf(buffer, sizeof(buffer), "not card found\r\n");
    pc.write(buffer, strlen(buffer));
    gOled2.clearDisplay();
    gOled2.setTextCursor(0, 0);
    gOled2.printf("card absence \r\n");
    gOled2.display();
    ThisThread::sleep_for(2s);

    continue;
}
```

圖 4.4.1、偵測現在是否有卡靠近 RFID

```
if (!RfChip.PICC_ReadCardSerial())
{
  ThisThread::sleep_for(1000ms);
  //pc.printf("card read\r\n");
  gOled2.clearDisplay();
  gOled2.setCursor(0, 0);
  gOled2.printf("card read\r\n");
  gOled2.display();
  snprintf(buffer, sizeof(buffer), "card read\r\n");
  pc.write(buffer, strlen(buffer));
  continue;
}
```

圖 4.4.2、讀取卡片 ID

4.4.2 OLED 顯示螢幕

本研究使用 SSD1306 作為 OLED 螢幕，使用透過 UART 與 STM32 連接，並且印出 RPI 回傳過來的訊息。其中函式庫修改自：

https://os.mbed.com/users/nkhorman/code/Adafruit_GFX/

同樣地，修改過的函式庫可見本專案儲存庫中 RC522 資料夾。

```
int16_t x = 0; // 起始 x 坐標
int16_t y = 0; // 起始 y 坐標
int16_t lineHeight = 8; // 每行文本的高度，根據文本大小調整

size_t start = 0;
size_t end = result.find('\n');
while (end != std::string::npos) {
    std::string line = result.substr(start, end - start);
    goled2.setTextCursor(x, y);
    goled2.printf("%s", line.c_str());
    y += lineHeight; // 移動到下一行
    start = end + 1;
    end = result.find('\n', start);
}

// 打印最後一行（如果有的話）
std::string line = result.substr(start);
if (!line.empty()) {
    goled2.setTextCursor(x, y);
    goled2.printf("%s", line.c_str());
}

goled2.display();
ThisThread::sleep_for(10s);
printf("Wait for next round: ");
}
```

圖 4.4.3、辨識成功時，STM32 將收到的訊息顯示在 OLED 螢幕上

```
else if (result.find("not 7 numbers") != std::string::npos){
    printf("ERROR: Recognize not 7 numbers\r\n");
    goled2.printf("not 7 numbers");
    goled2.display();
    ThisThread::sleep_for(10s);
}
else if (result.find("No entry record found for plate:") != std::string::npos){
    printf("ERROR: No entry record found for plate\r\n");
    goled2.printf("No entry record ");
    goled2.display();
    ThisThread::sleep_for(10s);
}
goled2.clearDisplay();
evqueue.call([&evqueue] { handleRFIDScan(evqueue); });
```

圖 4.4.4、當辨識失敗時，STM32 僅印出錯誤訊息至 OLED 上

4.4.3 伺服馬達控制閘門

本研究使用 SG90 伺服馬達以控制閘門的開關。此伺服馬達是藉由輸入 PWM 訊號的占空比，來控制馬達的角度。當設置占空比為 1 時能夠達到最大 180 度，而占空比為 0 時即為 0 度。程式實現的過程中，設定需要接收到 RPI 回傳辨識成功的訊號 “success” 才會控制馬達，若收到其他錯誤訊息時，則不會啟動馬達。

```
if (result.find("success") != std::string::npos){
    ....// 伺服馬達程式碼
    ....PwmOut PIN_SG90(PB_1);
    ....PIN_SG90.period(0.01f); // 週期為 10 毫秒 (100 Hz)
    ....//float open_duty = 1.0f;
    ....//float close_duty = 0.0f;

    ....for (int i = 0; i < 1; i++) {
    ....    // 增加 PWM 占空比，從 0.0 到 1.0
    ....    for (float duty_cycle = 0.0f; duty_cycle <= 1.0f; duty_cycle += 0.05f) {
    ....        PIN_SG90.write(duty_cycle); // 設置占空比
    ....        // printf("0->1\r\n");
    ....        ThisThread::sleep_for(20ms); // 延遲 100 毫秒
    ....    }
    ....    ThisThread::sleep_for(100ms);
    ....    // 減少 PWM 占空比，從 1.0 到 0.0
    ....    for (float duty_cycle = 1.0f; duty_cycle >= 0.0f; duty_cycle -= 0.05f) {
    ....        PIN_SG90.write(duty_cycle); // 設置占空比
    ....        // printf("1->0\r\n");
    ....        ThisThread::sleep_for(20ms); // 延遲 100 毫秒
    ....    }
    ....    ThisThread::sleep_for(100ms);
    ....}

    ....printf("result.c_str() = %s", result.c_str());
    ....// goled2.printf("%s\r\n", result.c_str());
}
```

圖 4.4.5、當收到 RPI 傳來辨識成功的訊號，才開啟閘門。

4.4.4 EVENT QUEUE

由於停車感應系統大部分時間是處於閒置狀態。因此本研究設定當感應到卡片時觸發中斷，而非使用輪巡偵測卡片，達到節省運算資源的效果。為了避免在執行中斷服務程式時，又被觸發中斷造成意外錯誤，因此本研究設計中斷服務程式用 event queue .call 的方式處理。每當觸發中斷時，在 queue 的後面新增一個待處理的事件，而不會立刻執行中斷服務程式，如此即可避免中斷被中斷的問題。

```
void handleRFIDScan(EventQueue& evqueue);  
void handleSocketDemo(EventQueue& evqueue);
```

圖 4.4.6、本研究使用到的兩種中斷事件

```
int main()  
{  
    EventQueue evqueue;  
    Thread evthread;  
    evthread.start(callback(&evqueue, &EventQueue::dispatch_forever));  
  
    //evqueue.call(handleRFIDScan);  
    //evqueue.call(handleSocketDemo);  
    evqueue.call([&evqueue] { handleRFIDScan(evqueue); });  
    // evqueue.call([&evqueue] { handleSocketDemo(evqueue); });  
    while(true){  
        ThisThread::sleep_for(osWaitForever);  
    }  
    //return 0;  
}
```

圖 4.4.7、在 main 當中設定 event queue dispatch forever

五、 結果討論

5.1 實驗結果

本研究的實驗結果除了在教室環境中以手繪車牌模擬實驗，也有至實際的停車場中，現場拍攝車牌即時推論。在教室環境中的入場測試可見此影片：

https://drive.google.com/file/d/1cHyW1G1ZNqEe9rGxsuYoBs.jpFF7JTz4W/view?usp=drive_link

其他測試的影片可見：

<https://drive.google.com/drive/folders/1aYixpH07C6Fz23jzy-IdXMyK0YIpHxc6>



圖 5.1、入場時只顯示車牌(左)離場時顯示車牌及進出場時間(右)

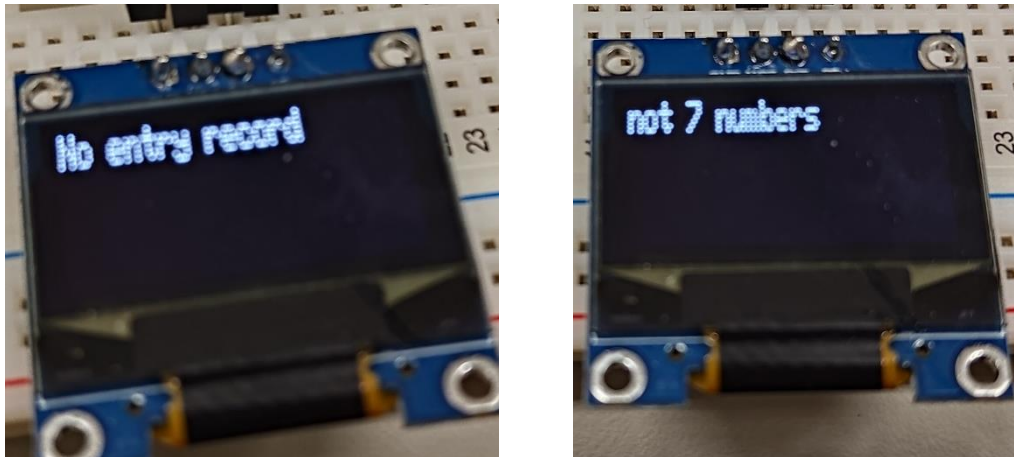


圖 5.2、辨識錯誤時顯示不同的錯誤訊息

5.2 未來展望

本研究已經實作了大部份車牌辨識停車管理系統所需的機能，但由於時間關係來不及實作 RFID 的儲值與付款功能。但因為已經實作出感應的機能，所以再多加儲值與付款功能應該不是太大的問題。

此外目前每次車牌辨識需要約 10 秒的時間，其中 5 秒是花費在 wifi 連接，而另外 5 秒花費在 yolo 車牌辨識。可以改善程式碼，讓 wifi 持續連線，並且使用更好的硬體和對 AI 模型做進一步的優化(如剪枝等)，以改善車牌辨識的速度。

最後本研究測試的環境，不像一般停車場可以固定攝影的角度、照明、背景等。如果可以固定這些條件，相信能夠得到更高的辨識準確度，也能減少一些預處理耗費的時間。

六、 參考資料

- https://github.com/adafruit/Adafruit_SSD1306
- https://os.mbed.com/teams/Project5_Software/code/RFID-RC522/docs/tip/