Automatic Guided Vehicle & Fleet Management System

作者：陳秉嘉、歐鎧豪

一、 目標說明

本專題要實作自動導引車（AGV，Automated Guided Vehicle）與其車隊管理系統（FMS，Fleet Management System），需要在指定場域中開發出 FMS 來對 AGV 下達指定的路由命令，讓 AGV 得以遵循指定的路徑行走，並即時將 AGV 之真實位置顯示於 FMS 上。本次所使用之場域中的賽道如下圖 1 所示：

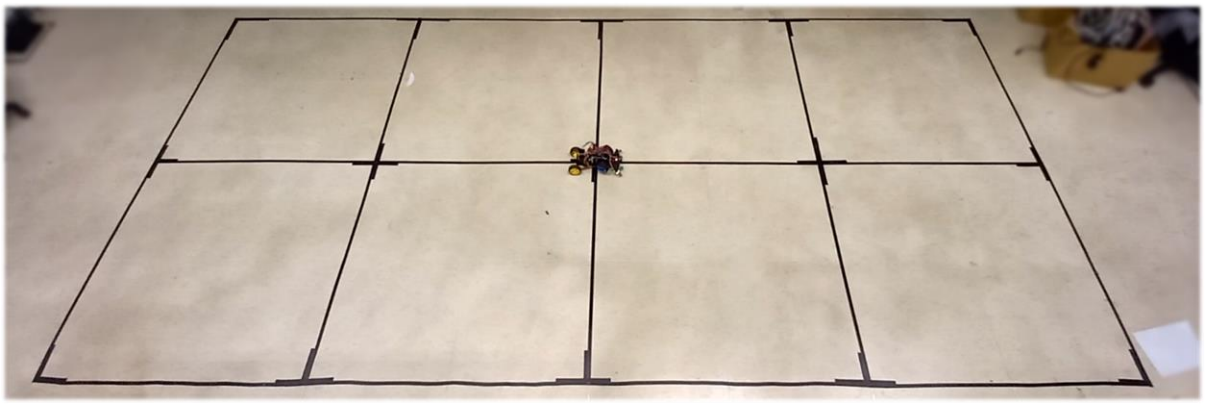


圖1 賽道俯視圖

為了模擬工廠中可能有多層的軌道同時在運輸（如晶圓廠中，負責載送矽晶圓到產線上的空中運輸車），我們將賽道分成三種軌跡，如下圖 2 所示，分別以紅、綠、藍三種顏色來代表不同的軌跡。對照圖 1 與圖 2 可知，兩者是相容的，我們會利用 FMS 來賦予軌道實質意義。

此外，為了模擬工廠內的 AGV 可能會有不同中繼站（如：電商的倉庫中，AGV 可能需要在不同的貨架上搬運商品），我們會在地圖中設立中繼站（如圖 2 中黑色方塊所示）來模擬上述的情況。

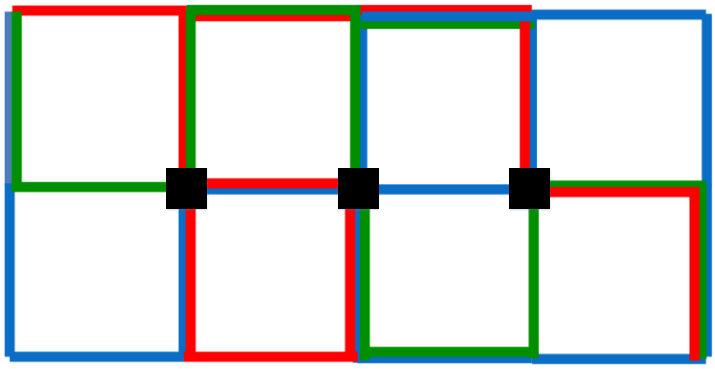


圖2 賽道分層軌跡

二、 硬體配置

1. 硬體材料

硬體材料清單如下表 1 所示，

表1 硬體材料清單

	Raspberry Pi 3 B+ ×1		電源轉接頭 ×1
	18650 鋰電池 ×4		雙槽 18650 電池盒 ×2
	馬達電源模組 ×1		三線直流電壓表 ×1
	直流馬達 ×2		輪子 ×2
	前導滾輪 ×1		紅外線循跡模組 ×5

2. 硬體介紹

(1) Raspberry Pi 3B+

Raspberry Pi 以下簡稱 Rpi，是基於 Linux 的單晶片電腦以其低價的硬體與自由的軟體被物聯網系統廣泛應用，可自由地選擇安裝作業系統，此外官方也提供免費的作業系統供下載使用，在此專案中以 Raspbian 作業系統為主，讀者也可選擇自己喜歡或擅長的作業系統來使用。

此外，大部分的情況下，我們會利用 SSH（Secure Shell Protocol）遠端登錄到 Rpi 來執行命令並以此控制 Rpi。若沒有特別設定，建議將個人電腦與 Rpi 連接上同一台無線路由器，讓它們處於相同的網域下，關於更詳細的指令與介紹請參照參考資料。

(2) 紅外線循跡模組

本專案採用 TCRT5000 紅外線尋跡感測模組，其中包含一組紅外線發射端與接收端。當發射的紅外線沒有被反射回來或是強度不夠時，其中的光敏三極管會處於關斷狀態，造成模組的輸出為低電位此時模組上的 LED 指示燈會呈現熄滅的狀態；相反地，若紅外線反射強度足夠時，會輸出高電位此時 LED 會被點亮，利用此特性我們可以用來偵測軌道以達成我們尋跡的目標，以下是尋跡模組的範例程式。

```
import RPi.GPIO as GPIO
import time
port = 7
GPIO.setmode(GPIO.BOARD)
GPIO.setup(port,GPIO.IN)
try:
    while True:
          print(GPIO.input(port))
          time.sleep(1)
except:
          GPIO.cleanup()
```

接著將模組上的 DO 腳位接至 Rpi 上的 7 號腳位並執行以上程式，就會在螢幕上印出目前模組的狀態。若為 1 的話代表前方沒有可吸收反射的物體，此時若將黑色或深色物體靠近他的接收端則會看到螢幕上顯示為 0；如果沒有改變的話可以調整模組上的靈敏度開關，藉由旋轉上面的螺絲來達到自己所要求的靈敏度。值得注意的是，Rpi 的 GPIO 有 BOARD 和 BCM 兩種模式，在此是使用 BOARD 模式，兩個模式間的腳位編號不同容易搞混。

(3) 馬達電源模組

本專案所使用的模組為 Adafruit 出版的 DC stepper motor hat，此模組為 Rpi 的擴充模組，其內部包含一個 12V 的升壓模組用來供應直流馬達以及一個 5V 的降壓模組作為 Rpi 的電源輸入，電路的區塊圖如下圖 3 所示：

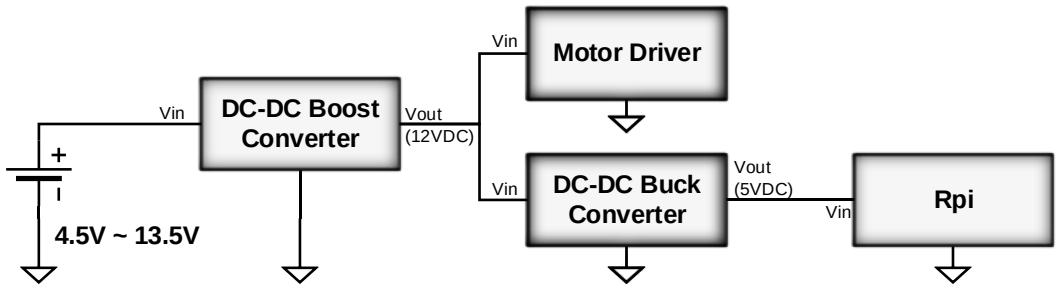


圖3 馬達電源模組電路圖

該模組在馬達驅動的部分，採用 TB6612 這顆 IC 作為驅動，它有著比 L298N 更強的驅動能力；並整合了 4 個控制埠，可同時控制兩個步進馬達或四個直流馬達。另外，此模組也擴充了許多電源輸出接口，可提供 5V 及 3.3V 輸出，在組裝時可做為紅外線尋跡模組的電源輸入；而關於如何控制直流馬達會在下一小節做介紹。

(4) 直流馬達

直流馬達主要作為 AGV 後輪驅動的動力來源，輸入電壓需求為 12V。在焊接電源線時須特別注意馬達上的正負極，若正負極相反時可能會造成馬達轉向相反，這時只需將電源正負調換即可。接著將電源接上馬達模組的控制埠上就可進行操控。控制埠有不同編號分別為 M1 到 M4，在此以 M1 做為示範範例，以下為控制直流馬達的範例程式：

```
import time
from adafruit_motorkit import MotorKit

kit = MotorKit()

kit.motor1.throttle = 1.0
time.sleep(0.5)
kit.motor1.throttle = 0
```

執行上述程式碼就可讓直流馬達全速轉動 0.5 秒，其中 `throttle` 的值可設為 0~1 的任意數，1 為全速轉動，0 為停止，藉此控制馬達的轉速。此外，也可將其設為-1~0，此時馬達會反轉，同理-1 為全速反轉，0 為停止。其中 `motor1` 代表是控制連接在控制埠編號 M1 上的直流馬達，若需同時控制多個馬達可將其改為 `motor1` ~ `motor4`。

(5) 前導滾輪

此滾輪我們將安裝在車體前端作為車子的前輪，透過該前導滾輪、配合兩個後輪的正反轉來做到以滾輪為支點的原地自轉，將轉彎半徑、迴轉半徑降到最小，更詳細的轉彎控制會在後面做介紹。

(6) 鋰電池與電池盒

本次專題使用並聯的兩組串聯兩顆 18650 鋰電池的電源（如下圖 4 所示）來做為整套 AGV 系統的電源輸入。由於當馬達啟動時，需要很大的啟動電流，以此並聯的方式作為電源供應，來增加可供給的最大電流，使得系統更加穩定。

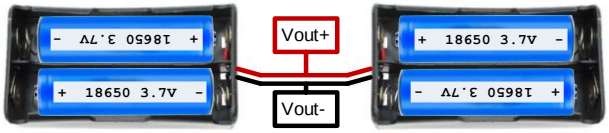


圖4 電源

(7) 三線直流電壓表

由於輸入電壓過高可能會使驅動模組燒毀，而電壓太低可能會使系統不能正常運作，透過連接該電壓表，得以即時監控電源電壓。值得注意的是，本專案使用的是三線式的直流電壓表監測模組，其與輸入電源的連接方法如下圖 5 所示。串聯的鋰電池最多可供應 DC8.4V，若輸入電壓低於 DV7V，建議為鋰電池充電後再繼續實驗。

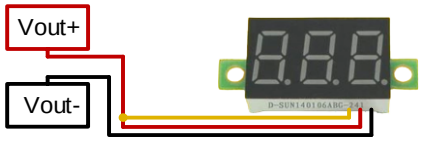


圖5 三線式電壓表連接圖

3. 硬體連接

此專題之詳細硬體接線圖如下圖 6 所示，首先把馬達驅動模組接到樹梅派的頭上，然後將直流馬達鎖到端子台上，接著將紅外線循跡模組連接至樹梅派，最後再插上電源即可。

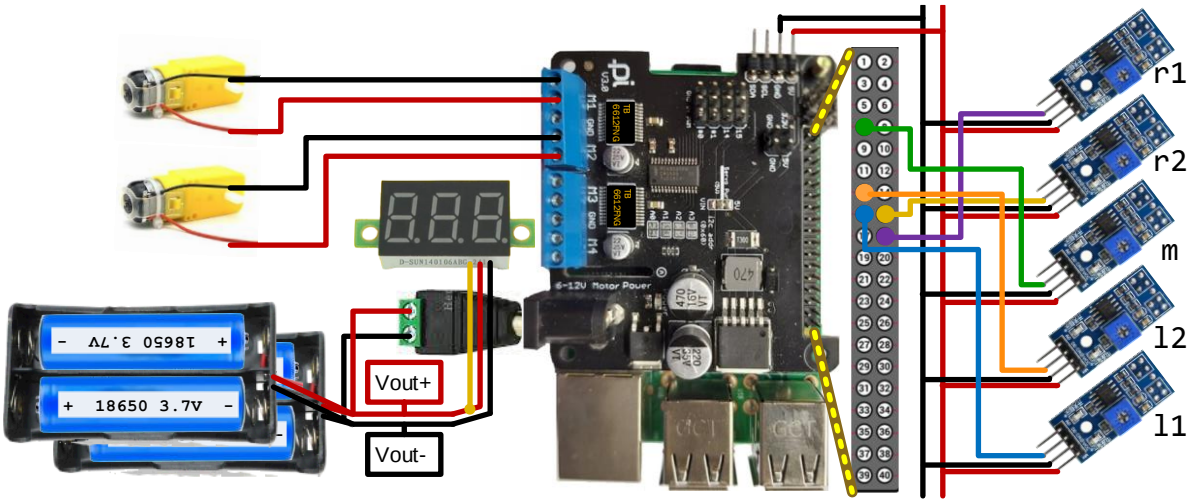


圖6 硬體配置圖

4. 硬體配置完成圖

本專題硬體配置完成圖如下圖 7~圖 9 所示，其分別為不同視角拍攝之外觀呈現。值得注意的是圖中的前輪（藍色的輪子）是沒有用到的，僅為當初實驗性質之配置。

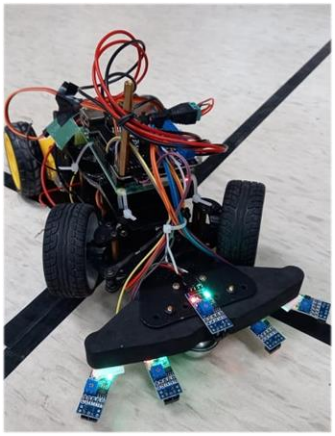


圖7 府視圖

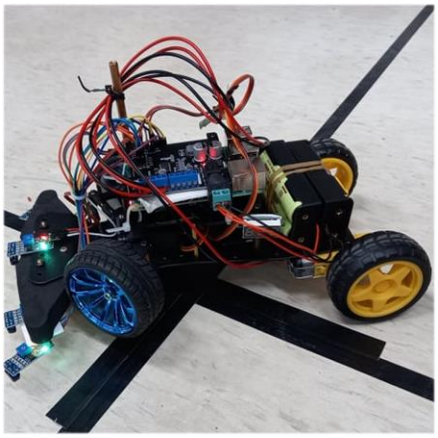


圖8 側視圖

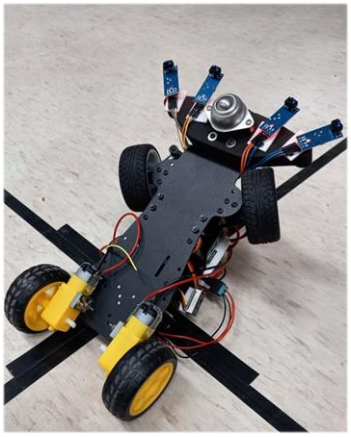


圖9 仰視圖

三、 系統架構說明

本專案之系統架構如下圖 10 所示，我們會以網頁的形式製作使用者的操作介面，該使用者操作介面會透過 JavaScript 向搭載在 Raspberry Pi 上的應用程式，以 HTTP 協定發送車輛行駛的路徑資訊。

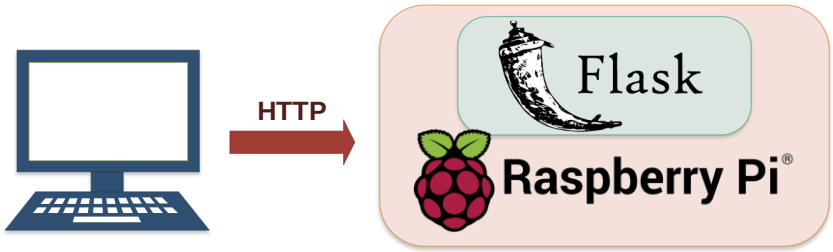


圖10 系統架構圖

我們搭載在 Raspberry Pi 上負責接收 HTTP 請求的應用程式係使用 Python 的 Flask 框架搭建。前端網頁會透過 HTTP 協定，傳送一串的路徑指令來命令車輛移動。其中，每個指令是由一個英文字母組成，該應用程式僅接受四種指令，分別是：

- F：向前移動，直到下一個路口停下來
- R：原地右轉
- L：原地左轉
- B：原地迴轉

例如，路徑指令「FRFLFBF」即代表「向前移動到路口停止；原地右轉；向前移動；原地左轉；向前移動；原地迴轉；向前移動」。所以，假設車輛的起始車頭是朝著右方，則車輛的行走軌跡會如下圖 11 所示。

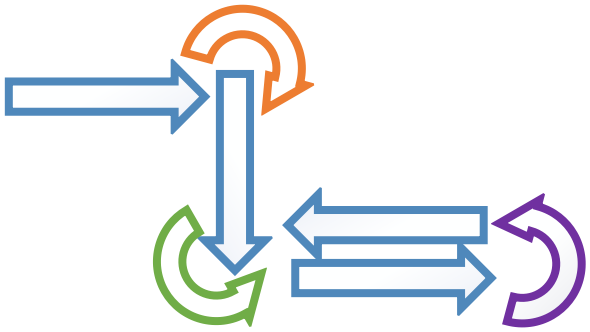


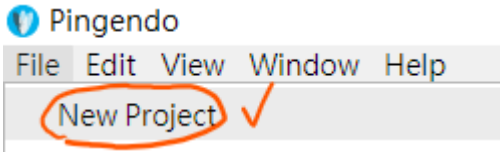
圖11 車輛行走軌跡範例

四、 前端使用者介面

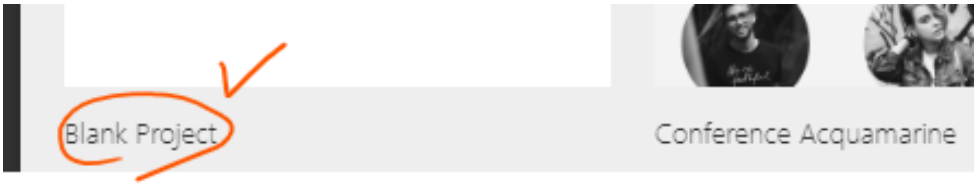
1. 以 Pingendo 搭建網頁雛形

本專案會使用 Pingendo 工具，讓開發者得以用拖拉的方式來建立網頁雛形。

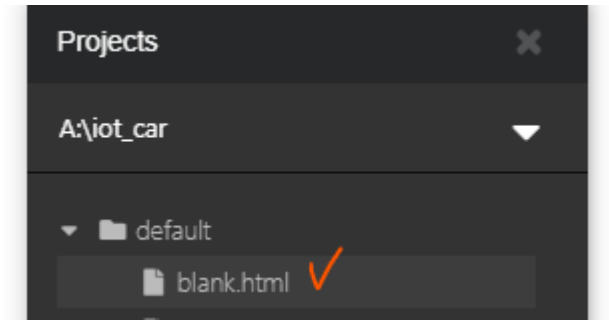
步驟1： 開啟新的專案，



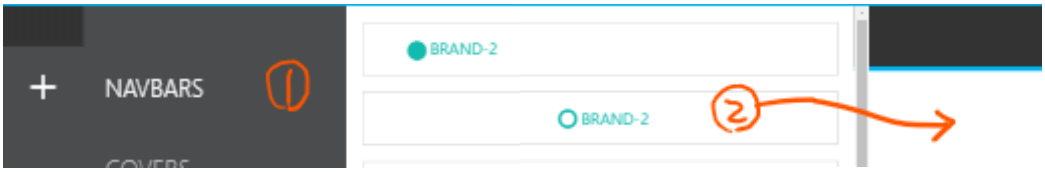
步驟2： 選擇空白的專案，並在下方選擇要儲存的地方，並按下 `Create` 按鈕



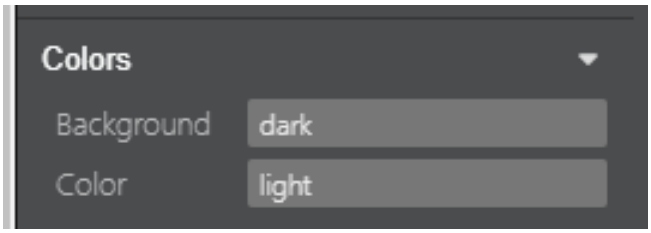
步驟3： 點開 blank.html 檔案



步驟4： 右方可以選擇喜歡的元件，將它拖曳到畫面中央

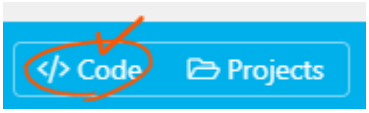


步驟5： 接著可以到右方面板更改我們想要的樣式



我們把 Background 改成 dark、把 Color 改成 light，效果如下

步驟6： 然而有些設定沒有提供選項可以點選，只能透過修改程式碼來達成效果。

點選右上角的 Code 按鈕  來修改程式碼。

找到下圖中的程式碼區塊，並將 `text-primary` 屬性刪除。

```
<div class="container d-flex justify-content-center"> <a class="navbar-brand text-primary" href="#">
    <i class="fa d-inline fa-lg fa-circle-o"></i>
    <b> BRAND-2</b>
  </a> </div>
```

步驟7： 接著拉出所需要的物件，完成圖如下圖 12，程式碼會在下頁中列出。

圖12 前端控制頁面雛形

透過 Pingendo 工具，以拖拉的方式建立頁面，並經過一些程式修改後，前端頁面雛形的程式碼如下列所示（礙於篇幅，僅列出重要的部分）：

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Access-Control-Allow-Origin" content="*">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" type="text/css">
  <link rel="stylesheet" href="theme.css" type="text/css">
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+TC&amp;display=swap" rel="stylesheet">
</head>
<body>
  <div class="py-5 text-center">
    <div class="container">
      <div class="row">
        <div class="text-center mx-auto col-lg-8">
          <form class="">
            <div class="form-group">
              <h4>Device IP Address</h4>
              <input type="text" id="ip" class="form-control" placeholder="Enter Device IP Address">
            </div>
          </form>
        </div>
      </div>
      <div class="row">
        <div class="text-center mx-auto col-lg-8">
          <ul class="nav nav-pills">
            <li class="nav-item">
              <a href="" id="R" class="active nav-link" data-toggle="pill" data-target="#tabone">紅色軌道</a>
            </li>
            <li class="nav-item">
              <a href="" id="G" class="nav-link" data-toggle="pill" data-target="#tabtwo">綠色軌道</a>
            </li>
            <li class="nav-item">
              <a href="" id="B" class="nav-link" data-toggle="pill" data-target="#tabthree">藍色軌道</a>
            </li>
          </ul>
          <div class="tab-content mt-2">
            <div class="tab-pane fade show active" id="tabone" role="tabpanel">
              <img class="img-fluid d-block" src="https://static.pingendo.com/cover-bubble-light.svg">
            </div>
            <div class="tab-pane fade" id="tabtwo" role="tabpanel">
              <img class="img-fluid d-block" src="https://static.pingendo.com/cover-bubble-light.svg">
            </div>
            <div class="tab-pane fade" id="tabthree" role="tabpanel">
              <img class="img-fluid d-block" src="https://static.pingendo.com/cover-bubble-light.svg">
            </div>
          </div>
          <a class="btn btn-outline-dark" id="go_btn">  GO!  </a>
        </div>
      </div>
    </div>
  </div>
```

```
54    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
55    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
56      integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
57      crossorigin="anonymous"></script>
58    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
59      integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
60      crossorigin="anonymous"></script>
61  </body>
62  </html>
```

2. 以 HTML5 的 Canvas 繪製地圖

canvas 是 HTML5 提供的畫布元素，我們可以利用 JavaScript 在這個元素上繪製圖形。我們想要將地圖視覺化地繪製在網頁中，所以必須在網頁中放入 canvas 元素，分別將上面程式碼中的第 40、43、46 行以下列的程式碼替換：

```
<canvas id="cv1" class="img-fluid d-block" width="1600" height="900"></canvas>
<canvas id="cv2" class="img-fluid d-block" width="1600" height="900"></canvas>
<canvas id="cv3" class="img-fluid d-block" width="1600" height="900"></canvas>
```

修改完畢後，我們將使用大量的 JavaScript 來繪製我們想要的圖形。首先，要使用下列的程式碼來取得上述之 canvas 元素

```
// canvas 1 (紅色軌道)
var cv1 = document.getElementById("cv1");
var cv1_ctx = cv1.getContext("2d");
// canvas 2 (綠色軌道)
var cv2 = document.getElementById("cv2");
var cv2_ctx = cv2.getContext("2d");
// canvas 3 (藍色軌道)
var cv3 = document.getElementById("cv3");
var cv3_ctx = cv3.getContext("2d");
```

接下來我們將分別敘述如何在 canvas 元素中畫線、畫圓、畫方塊，以及如何偵測畫布上的滑鼠點擊事件和取得點擊時的座標，綜合上述的技術，讀者應當有能力達成我們想要的目標，即在不同的頁籤中顯示設定好的地圖路徑，如下圖 13 所示。

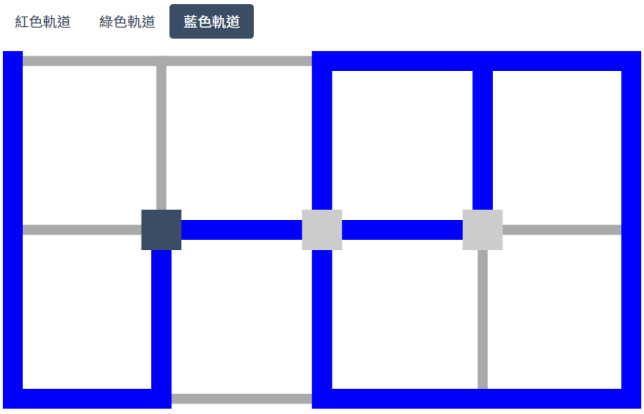


圖13 選定頁籤後，可以在畫布中顯示對應的路徑（此以藍色的路徑為例）

(1) 繪圖原理介紹

我們在螢幕上所看到的所有圖像其實都是由許多包含不同亮度的 RGB 小光點組成，如圖 14 所示。而這些小光點可以呈現出各種顏色，經過適當排列後的小光點即組合出我們所看到的圖形。

數學坐標系的 X 軸為向右遞增、Y 軸則為向上遞增。而在電腦程式中 X 軸仍為向右遞增，Y 座標則是向下遞增，且(X,Y)皆須為整數，如下圖 15 所示。若想要產生一條從(2,1)到(8,9)的紫色線條，只需計算會經過的像素，然後將這些像素點改成指定顏色即可，如圖 16 所示。

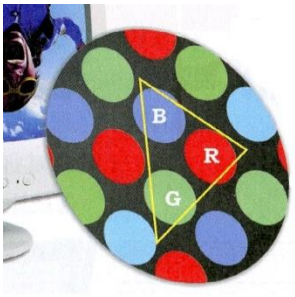


圖14 螢幕成像原理

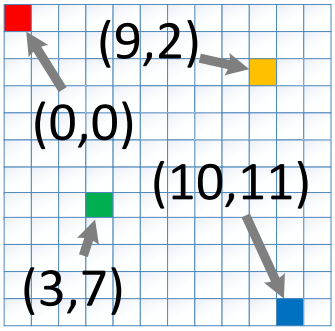


圖15 電腦程式座標表示

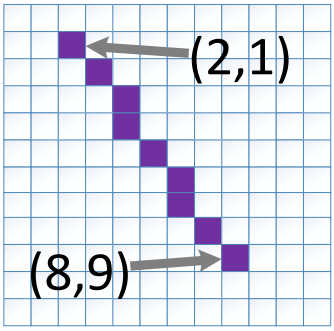


圖16 圖形成像原理

(2) canvas 元素中繪製直線

在我們的前端控制頁面中，會用會使用繪製直線的方式來描繪軌道。想要在畫布中繪製出一條直線，我們所需執行的動作為：取得繪圖元件、設定線條寬度與顏色、指定繪製路徑。其範例程式如下列所示：

```
var cv = document.getElementById("cv1"); // 取得畫布元素
var cv_ctx = cv.getContext("2d"); // 取得該畫布之 2D 繪圖物件物件
cv_ctx.lineWidth = 10; // 設定線條寬度
cv_ctx.strokeStyle = '#64A600'; // 設定線條顏色
cv_ctx.beginPath(); // 產生一個新路徑，產生後再使用繪圖指令來設定路徑
cv_ctx.moveTo(5, 5); // 設定繪圖起始點
cv_ctx.lineTo(500, 50); // 從目前繪圖點，畫一條直線到指定的座標點
cv_ctx.stroke(); // 畫出圖形的邊框
```

上述程式會從畫布中的座標(5,5)到座標(500,50)繪製出一條直線，執行範例如下圖 17 所示：

圖17 直線繪製示範

(3) canvas 元素中繪製矩形

在我們的前端控制頁面中，會用會使用繪製矩形的方式來描繪中繼站點。想要在畫布中繪製矩形，我們所需執行的動作為：取得繪圖元件、設定顏色、指定矩形的繪製座標及寬高。其範例程式如下列所示：

```
var cv = document.getElementById("cv1"); // 取得畫布元素
var cv_ctx = cv.getContext("2d"); // 取得該畫布之 2D 繪圖物件物件
cv_ctx.fillStyle = "#64A600"; // 設定圖形要填滿的顏色
cv_ctx.fillRect(20, 10, 200, 100); // 繪製填滿的矩形，參數為座標與寬、高
```

上述程式會以座標(20,10)為矩形之左上角的起始點，並向右下方繪製出寬 200 高 100 的矩形，範例執行結果如下圖 18 所示：

圖18 方塊繪製示範

(4) canvas 元素中繪製圓形

在我們的前端控制頁面中，會用會使用繪製圓形的方式來描繪目前車輛的位置。想要在畫布中繪製圓形，我們所需執行的動作為：取得繪圖元件、執行圓形的繪製指令，設定顏色並填滿它。其範例程式如下列所示：

```
var cv = document.getElementById("cv2"); // 取得畫布元素
var cv_ctx = cv.getContext("2d"); // 取得該畫布之 2D 繪圖物件物件
cv_ctx.arc(500, 100, 50, 0, 2 * Math.PI, true); // x, y, r, startAngle, endAngle, clockwise
cv_ctx.fillStyle = "#B7770D"; // 設定填滿的顏色
cv_ctx.fill(); // 將圖形填滿
```

上述程式會以座標(500,100)為圓心、以 50 為半徑，繪製一個圓形（逆時針，角度 0~2π ），範例執行結果如下圖 19 所示：

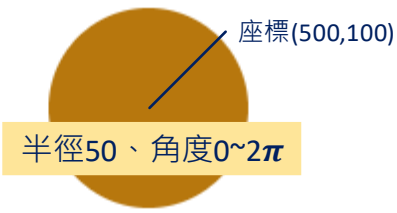


圖19 方塊繪製示範

(5) 軌道繪製程序

繪圖的程序比較繁瑣，但是並不困難；經由上述的介紹，相信各位一定有能力將軌道繪製完成，底下僅以概念式的方式介紹軌道的繪製流程，並不會列出完整的程式碼。首先，我們先以線條的方式繪製出如圖 20 中所展示的格線作為軌道的底圖。

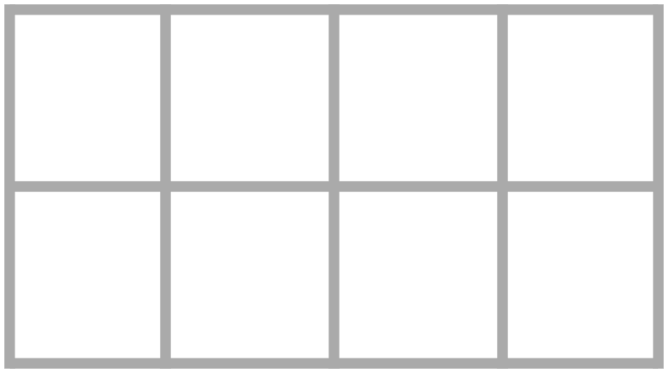


圖20 軌道底圖

接著，我們仍然是繼續使用直線的方式，繪製出特定顏色的路徑，下圖 21 以紅色的軌道路徑作為示範：

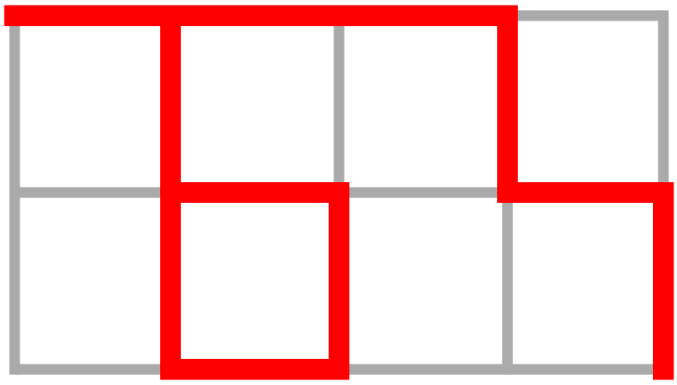


圖21 繪製軌道路徑

最後，我們會以繪製矩形的方式，在規定的地點上繪製出中繼站，如下圖 22 所示。

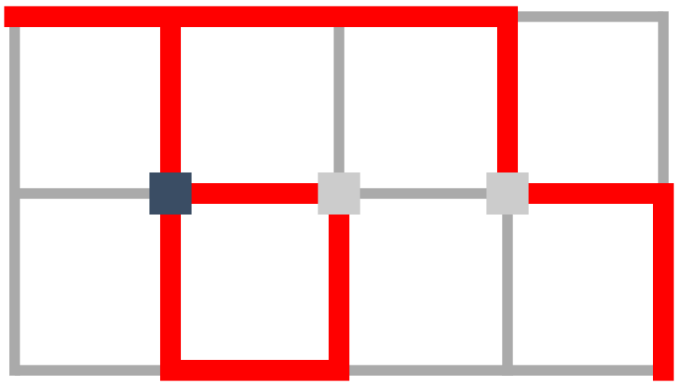


圖22 繪製中繼站

綜合上述繪製程序，我們可以定義出 `drawCheckpoint()`、`drawMap()`兩個函式（如下列程式碼所示），應用前面所提到的技術，相信各位一定有能力自行將程式中的 `TODO 1` ~ `TODO 4` 完成。

```
function drawCheckpoint(canvas, cv_ctx, checkpoint_size) {
  // TODO 3: 清除中繼站的顏色
  // TODO 4: 點選到的中繼站上色
}
function drawMap() {
  cv1_ctx.clearRect(0, 0, cv1.width, cv1.height);
  // TODO 1: 畫出底圖的軌道
  // TODO 2: 畫出顏色的軌道
  drawCheckpoint(cv1, cv1_ctx, checkpoint_size);
}
drawMap()
```

(6) canvas 元素中偵測畫布上的滑鼠點擊事件和取得點擊時的座標

我們想要透過監聽 canvas 畫布元素的點擊事件、取得並判斷點擊時的滑鼠位置，來達到中繼點選擇的效果。而監聽 canvas 點擊事件的方法與監聽其他 HTML 元素相同，範例程式碼如下列所示：

```
var checkpoint = 1  // {1, 2, 3}
cv1.addEventListener('click', (e) => {
  var pos = getMousePos(cv1, e); // 取得滑鼠的座標
  if (滑鼠點選到中繼點 1 的範圍內) {
    checkpoint = 1 // 設定中繼站編號
  }
  if (滑鼠點選到中繼點 2 的範圍內) {
    checkpoint = 2 // 設定中繼站編號
  }
  if (滑鼠點選到中繼點 3 的範圍內) {
    checkpoint = 3 // 設定中繼站編號
  }
  drawMap()
});
```

由於滑鼠位於畫布上的座標計算並不是那麼直覺，因此我們特地將取得滑鼠座標的功能獨立撰寫成 `getMousePos()`函式，如下列程式碼所示：（資料來源列於文末）

```
function getMousePos(canvas, evt) {
  var rect = canvas.getBoundingClientRect(), // abs. size of element
    scaleX = canvas.width / rect.width,    // relationship bitmap vs. element for X
    scaleY = canvas.height / rect.height;  // relationship bitmap vs. element for Y
  return {
    x: (evt.clientX - rect.left) * scaleX,   // scale mouse coordinates after they have
    y: (evt.clientY - rect.top) * scaleY     // been adjusted to be relative to element
  }
}
```

3. 發送車輛路徑指令到 Raspberry Pi 上

我們需要向搭載在 Raspberry Pi 上的控制程式發出 HTTP 請求，為了方便起見，我們先定義了一個取得網址的函式（如下列所示），該函式會透過網頁上的輸入框取得使用者指定之車輛控制板的 IP 位置，並與協定、埠號做文字串接。

```
function GetURL() {
  var ip = $('#ip').val();
  url = 'http://' + ip + ':8080'
  return url
}
```

我們先假定車載系統上具有兩隻 API，分別是 `GET /reset` 與 `POST /map`。其中 `GET /reset` 會重設車輛的位置與方向於初始的起點，而 `POST /map` 則負責接收路徑指令並執行對應的動作。當我們按下網頁中 `GO!` 的按鈕，我們會執行的動作依序為：重設車輛狀態、準備路徑資訊、發送路徑資訊。實作程式碼如下列所示：

```
$("#go_btn").click(function () {
  $.ajax({
    crossDomain: true,
    url: GetURL() + '/reset',
    method: "GET",
    cache: false,
    success: function (response) {
      track = $(".active.nav-link").attr("id"); // {R, G, B}
      var path = "";
      if (track == 'R') {
        if (checkpoint == 1) {
          path = "FRFBFRFFRFLFRF"
        } else if (checkpoint == 2) {
          path = "FRFLFBFRFRFFRFLFRF"
        } else if (checkpoint == 3) {
          path = "FFFRFLFRF"
        }
      } --- 省略路徑 G、B 的程式碼 ---
      $.ajax({
        url: GetURL() + '/map',
        method: "POST",
        data: path,
        contentType: "application/json",
        crossDomain: true,
        cache: false,
        success: function (response) {
          console.log(response)
        },
        error: function () {
        }
      });
    },
    error: function () {
    }
  });
});
```

4. 獲取車輛位置並在網頁中顯示

我們希望能夠從車載系統上取得目前車輛的位置，並顯示於我們的前端網頁中，該車輛的位置會以閃爍的黃點來代表。我們先假定車載系統上具有 `GET /location` 這隻 API，它會回傳目前車輛所在的位置(x, y)。我們會不斷地呼叫 `setTimeout()` 來達到週期性更新畫面與車輛位置的目的。程式碼如下列所示：

```
var car_color_switch = 0;
var car_update_interval = 800;
function DrawCar() {
  $.ajax({
    crossDomain: true,
    url: GetURL() + '/location',
    method: "GET",
    cache: false,
    success: function (response) {
      console.log(response) // (x, y)

      drawMap()

      track = $(".active.nav-link").attr("id"); // {R, G, B}
      if (track == 'R') {
        pos_x = cv1.width / 4 * response.X
        pos_y = cv1.height / 2 * response.Y

        cv1_ctx.beginPath();
        cv1_ctx.arc(pos_x, pos_y, 50, 0, 2 * Math.PI, true);
        if (car_color_switch == 0) {
          cv1_ctx.fillStyle = "#B7770D";
          car_color_switch = 1;
        } else {
          cv1_ctx.fillStyle = "#F7BC5B";
          car_color_switch = 0;
        }
        cv1_ctx.fill();
      }--- 省略路徑 G、B 的程式碼 ---
      setTimeout(DrawCar, car_update_interval);
    },
    error: function () {
      setTimeout(DrawCar, car_update_interval);
    }
  });
}
$(document).ready(function () {
  setTimeout(DrawCar, car_update_interval);
});
```

五、 後端系統開發

1. 先備知識 – 旋轉矩陣（Rotation Matrix）

假設在直角坐標系中有一個向量(x,y)，我們想要以原點(0,0)為圓心，逆時針方向旋轉θ角度，則新的座標(x’,y’)可以用下列公式取得：

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

以矩陣的方式表示為：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

2. API 介面定義

以下列出我們車載系統的 API 介面定義及功能說明：

- `GET /init`

  清空並初始化車載系統上的資料庫，其資料庫內儲存了目前車輛所在之位置座標(X,Y)及車頭的方向向量(DX,DY)。

- `GET /reset`

  重設車輛所在位置(X,Y)為(0,0)、重設車頭方向向量(DX,DY)為(1,0)。

- `GET /location`

  取得車輛所在之位置座標(X,Y)及車頭的方向向量(DX,DY)。

- `POST /map`

  接收並執行路徑行駛的指令字串。

3. API 功能實作

(1) 設定專案

在開始專案功能撰寫之前，我們必須先引入一些函式庫，並對專案做出一些必要設定（如下列程式碼所示）。值得注意的是，為了讓跨來源資源共用（CORS）可行，我們有執行`CORS()`指令，目的是讓該應用程式得以允許不同域名的存取。

```
from flask import Flask, jsonify, request
from flask_cors import CORS
import numpy as np
import sqlite3

app = Flask(__name__)
CORS(app)
```

(2) 資料庫存取相關函式

A. 資料庫初始化 API

此小節將實作 `GET /init` 的 API 程式，首先會先初始化`'iot5.db'`，接著會新建 `device` 表格（用來存放車輛目前的位置及車頭朝向），最後會插入一筆初始資料（車輛所在位置(X,Y)設為(0,0)、車頭朝向(DX,DY)設為(1,0)）。實作程式碼如下：

```
@app.route('/init', methods=['GET'])
# 初始化資料庫
def init():
    conn = sqlite3.connect('iot5.db')
    c = conn.cursor()

    # 清除表格
    c.execute('''drop table if exists device;''')
    # 建立表格
    c.execute('''create table device
       (ID    INTEGER    PRIMARY KEY    AUTOINCREMENT,
        X     INTEGER                   NOT NULL,
        Y     INTEGER                   NOT NULL,
        DX    INTEGER                   NOT NULL,
        DY    INTEGER                   NOT NULL);''')
    # 新增初始位置
    c.execute("insert into device (X,Y,DX,DY) values (0,0,1,0);")

    conn.commit()
    conn.close()
    return jsonify("OK")
```

B. 車輛位置及車頭朝向更新函式

此小節將新增一個函式，用來更新車輛位置及車頭朝向，實作程式碼如下：

```
def updateDevice(X, Y, DX, DY):
    conn = sqlite3.connect('iot5.db')
    c = conn.cursor()

    c.execute(" UPDATE device SET X=?, Y=?, DX=?, DY=? WHERE ID = 1;",
              (X, Y, DX, DY))

    conn.commit()
    conn.close()
```

C. 重設車輛位置及車頭方向 API

此小節將實作 `GET /reset` 的 API 程式，它會呼叫道上一段程式碼來去設定車輛位置及車頭朝向，實作程式碼如下：

```
@app.route('/reset', methods=['GET'])
# 重設 rpi 目前的位置為(0,0)
# 需要有 global var 去紀錄
def reset():
    updateDevice(0, 0, 1, 0)
    return jsonify("OK")
```

D. 取得車輛位置及車頭方向 API

此小節將實作 `GET /location` 的 API 程式，它會從資料庫中取出目前的車輛位置及車頭朝向，實作程式碼如下：

```
@app.route('/location', methods=['GET'])
# 取得 rpi 目前的位置
# 需要有 global var 去紀錄
def location():
    conn = sqlite3.connect('iot5.db')
    c = conn.cursor()
    c.execute(" SELECT X,Y,DX,DY FROM device WHERE ID = 1;")
    obj = c.fetchone()
    conn.close()
    return jsonify({"X": obj[0], "Y": obj[1], "DX": obj[2], "DY": obj[3]})
```

(3) 路徑接收並行駛

A. 宣告車輛行駛之函式介面

這邊我們先宣告能夠使車輛執行指令的函式，以便我們待會使用：

```
def forward():
    pass

def turn_left():
    pass

def turn_right():
    pass

def turn_around():
    pass
```

B. 車輛行走主函式

此小節將實作解析指令並讓車輛行走的主要程式。在行走的同時，我們會同步更新資料庫中的車輛位置及車頭朝向的紀錄。首先，我們會先從資料庫中取得車輛的位置及朝向，接著會迭代指令字串中的每個指令字元，分述如下：

- 若為向前走(`'F'`)，則執行向前走的 `forward()` 函式，並令車輛位置加上車頭朝向的單位向量。
- 若為左轉(`'L'`)或右轉(`'R'`)，則執行 `turn_left()` 或 `turn_right()` 函式，並將車頭朝向乘上旋轉矩陣。
- 若為迴轉(`'B'`)，則執行 `turn_around()` 函式，並將車頭朝向直接乘以 `-1` 。

實作程式碼如下列所示：

```
def drive(cmd):
    conn = sqlite3.connect('iot5.db')
    c = conn.cursor()
    c.execute(" SELECT X,Y,DX,DY FROM device WHERE ID = 1;")
    obj = c.fetchone()
    conn.close()
    pos = np.array([obj[0], obj[1]])
    dir = np.array([obj[2], obj[3]])

    for c in cmd:
        if c == 'F':
            forward()  # 呼叫車車向前
            pos += dir
        elif c == 'R':
            turn_right()  # 呼叫車車右轉
            c, s = np.cos(np.radians(-90)), np.sin(np.radians(-90))
            rotation_matrix = np.array(((c, -s), (s, c)))
            dir = np.round(dir.dot(rotation_matrix)).astype(int)
        elif c == 'L':
            turn_left()  # 呼叫車車左轉
            c, s = np.cos(np.radians(90)), np.sin(np.radians(90))
            rotation_matrix = np.array(((c, -s), (s, c)))
            dir = np.round(dir.dot(rotation_matrix)).astype(int)
        elif c == 'B':
            turn_around()  # 呼叫車車迴轉
            dir *= -1
        updateDevice(pos[0].item(), pos[1].item(), dir[0].item(), dir[1].item())
```

C. 接收前端網頁呼叫指令之 API

此小節將實作 `POST /map` 的 API 程式，它透過將請求中的路徑字串傳遞到上述的 `drive()` 函式中實現，實作程式碼如下：

```
@app.route('/map', methods=['POST'])
# 發送路由到 rpi 上
# F: 往前一單位
# R: 右轉
# L: 左轉
# B: 迴轉
def map():
    cmd = request.data.decode("utf-8")
    drive(cmd)
    return "done!"
```

(4) 程式執行

我們要在 Python 程式腳本的尾端加上 Flask 框架的啟動程式碼，如下列所示：

```
if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=8080)
```

最後，將整個 Python 腳本儲存成 `app.py`，然後在終端機以 `python3 app.py` 執行我們撰寫好的車載應用程式。

六、 車輛驅動

首先我們先對 Rpi 的 GPIO 與馬達模組做一些初始化的配置，程式碼如下列所示：

```
import RPi.GPIO as GPIO
from adafruit_motorkit import MotorKit

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
r1 = 18
r2 = 16
m = 7
l2 = 13
l1 = 15
GPIO.setup(r1,GPIO.IN)
GPIO.setup(r2,GPIO.IN)
GPIO.setup(m,GPIO.IN)
GPIO.setup(l1,GPIO.IN)
GPIO.setup(l2,GPIO.IN)

kit = MotorKit()
```

以上主要是設定紅外線循跡模組的訊號輸入和馬達模組的應用套件，而紅外線模組的編號與圖 6 相同，`r1` ~ `l1` 分別是從最右邊到最左邊的紅外線模組。當然 GPIO 的輸入埠（Input Port）可以依個人情況來做調整，但須注意有些特定功能的 port 不可混用，在此只是作為範例使用。

接著我們將繼續實現在後端系統中尚未實作的 `forward()`、`turn_left()`、`turn_right()`、`turn_around()`函式，將馬達控制、循跡功能與後端系統做整合。首先，由於我們需要控制後輪的兩顆直流馬達來達成前進、後退與轉向的動作，因此需要一個函數如下：

```
def motor(left_speed,right_speed,s_time=0.01):
    kit.motor1.throttle = right_speed
    kit.motor2.throttle = left_speed
    time.sleep(s_time)
    kit.motor1.throttle = 0
    kit.motor2.throttle = 0
```

此函式可以分別控制左輪、右輪的轉速（`left_speed`、`right_speed` 參數）與持續運轉的時間長度(`delay time` 參數，以秒為單位)，來達到前進、後退與轉彎的功能；而當偵測到車輛偏移軌道時，也可以使用該函式來進行修正。

接著將正式來實作前進（`forward()`）、原地左轉（`turn_left()`）、原地右轉（`turn_right()`）、原地迴轉（`turn_around()`）四個功能函數。

1. 前進（forward()）

forward()函式一開始會先前進一小段距離，因為若連續收到兩個 forward()指令，可能會造成它還陷在路口中而沒辦法繼續行駛。接著，程式的判斷邏輯分成四種情況，分述如下：

（1）最左邊或最右邊的紅外線循跡模組（圖 6 中的 l1 或 r1）偵測到黑線，代表車輛已行進到路口，必須停下來等待下一個指令。

（2）若車輛處於黑色軌道上（圖 6 中的 l2 與 r2 皆無偵測到黑線）則繼續往前進。

（3）偵測到車輛偏右（圖 6 中的 l2 偵測到黑線）時，即利用兩輪的轉速差來修正。

（4）偵測到車輛偏左（圖 6 中的 r2 偵測到黑線）時，即利用兩輪的轉速差來修正。

```
def forward():
    motor(0.75,0.75,0.15)
    while True:
        if (GPIO.input(r1)==1) or (GPIO.input(l1)==1):
            motor(0,0)
            time.sleep(0.2)
            return True
        if GPIO.input(l2)==0 and GPIO.input(r2)==0:     # 直行
            motor(0.75,0.75)
        if GPIO.input(l2)==1 and GPIO.input(r2)==0:     # 偏右
            motor(0,1,0.02)
        if GPIO.input(l2)==0 and GPIO.input(r2)==1:     # 偏左
            motor(1,0,0.02)
```

2. 原地左轉（turn_left()）

我們會利用左右輪以相反方向旋轉來達到轉向的目的。由於我們的紅外線循跡模組配置在凸出車頭的最前方，所以一開始會先前進一小段，讓旋轉後的車體得以落在軌道上。接著就讓車體開始旋轉；由於不同地面材質、不同輪胎摩擦係數不同，所以旋轉的持續時間必須依實際環境狀況去做校正。而後面的四個判斷式是用來微調修正車輛轉過頭或是轉不夠的情況，畢竟每個路口的粗糙程度不一，很難以一個參數而貫之。

```
def turn_left():
    motor(0.75,0.75,0.08)
    time.sleep(0.1)
    motor(-1,1,1.5)
    time.sleep(0.1)
    if GPIO.input(r2)==1:
        motor(0.75,0.25,0.1)
    if GPIO.input(l2)==1:
        motor(0.25,0.75,0.1)
    if GPIO.input(r1)==1:
        motor(1,0,0.1)
    if GPIO.input(l1)==1:
        motor(0,1,0.1)
    return True
```

3. 原地右轉（`turn_right()`）

原地右轉的旋轉原理與方式與前一小節類似，在此不再贅述，僅列出參考程式碼如下列所示：

```
def turn_right():
    motor(0.75,0.75,0.08)
    time.sleep(0.1)
    motor(1,-1,1.5)
    time.sleep(0.1)
    if GPIO.input(r2)==1:
        motor(0.75,0.25,0.1)
    if GPIO.input(l2)==1:
        motor(0.25,0.75,0.1)
    if GPIO.input(r1)==1:
        motor(1,0,0.1)
    if GPIO.input(l1)==1:
        motor(0,1,0.1)
    return True
```

4. 原地迴轉（`turn_around()`）

原地迴轉的原理與程式說明與左右轉相同，理論上將轉彎的時間長度設為兩倍即可完成迴轉，但實測後發現會需要設為更久的時間。迴轉是這四個函式中最難實做的，需要花多一些時間來做參數的微調校正，而後面的修正項也必須更加小心才可以完成漂亮的迴轉而不超出軌道。

```
def turn_around():
    motor(0.5,0.5)
    time.sleep(0.1)
    motor(-1,1,3.3)
    time.sleep(0.1)
    if GPIO.input(r2)==1:
        motor(0.75,0.25,0.1)
    if GPIO.input(l2)==1:
        motor(0.25,0.75,0.1)
    if GPIO.input(r1)==1:
        motor(1,-0.2,0.3)
    if GPIO.input(l1)==1:
        motor(-0.2,1,0.3)
    return True
```

七、 結論與未來展望

本專案成功實做了 FMS 與 AGV 的基礎功能，透過友善的使用者介面可以輕鬆地對車輛下達路由指令，並獲得車輛的即時位置資訊。而這樣的車輛構造配置，經實際校正與測試後可以非常流暢地行駛。透過上述所介紹的技術，未來若要擴充成多輛 AGV 的車隊也非常容易，真正地實現了 AGV Fleet Management 的主要精神與功能。

八、 成果展示

本專案之成果展示影片如下列網址所示：

https://youtu.be/i42_GLMrgAw

本專案之完整程式碼公開於以下程式碼儲存庫：

https://github.com/okh8609/IOT_Project5_AGV-and-FMS

九、 參考資料

- "Drawing shapes with canvas."
  https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes
- "Real mouse position in canvas."
  https://stackoverflow.com/questions/17130395/real-mouse-position-in-canvas