

龍 華 科 技 大 學

機 械 工 程 系

專題製作報告

無人運輸生產線

Unmanned transportation production line

製作學生：

D1064113011 王暉翰

D1064113023 楊明翰

D1064113043 歐陽承岳

指導老師： 陳詩豐 副教授

中 華 民 國 一 零 九 年 六 月

摘要

近年來由於人類工業科技的提升，眾多工業科技的市場趨向逐漸往無人化之方向發展，加上機械手臂的普及，使得無人運輸系統變得相當地熱門，為提升自身競爭力並精進於機電整合領域之學識，故而選擇探討與鑽研無人化工廠以機器人取代生產之重要性。

本專題主要利用機械手臂以及網路攝影機鏡頭，來實現使用機械視覺配合機械手臂進行物料夾取動作。並使用邏輯電路開發版（Arduino）及各式感測器來操控並自行製作簡易無人自動導引載具（AGV，以下簡稱自走車）。而產線設計的部分，則是使用單晶片電腦（Raspberry Pi）來控制無人運輸系統以達到產線整合之目的。

在本次專題製作中不論是軟體還是硬體部分都遭遇到諸多問題，比如軟體、硬體的安裝和更新，還有自走車及攝影機腳架的機構設計，各個感測器的電路分配及穩定性，這些都是干涉到整體產線運行流暢度的因素，需要一一克服。

Abstract

In recent years, due to the improvement of human industrial technology , The market trend of many industrial technologies is gradually developing towards unmanned direction , and the popularity of robotic arms , those making unmanned transportation systems quite popular , to enhance our competitiveness and improve our knowledge in the field of electromechanical integration , Therefore , we chose to explore and study the importance of unmanned factories replace production with robots 。

This thesis mainly uses the robotic arm and webcam lens to realize the use of mechanical vision and mechanical arm for material clamping and use the Single-board microcontroller 、 various sensors to control and make simple unmanned automatic guidance vehicle 。

In the part of the production line design , we use Raspberry Pi to control the unmanned transportation system to achieve the purpose of production line integration 。

In this thesis, both software and hardware have encountered many problems such as software and hardware installation and update 、 AGV and webcam's tripod mechanism design 、 circuit distribution and stability of each sensor , these are factors that interfere with the smoothness of the overall production line , need to overcome one by one 。

誌謝

在本次專題報告之前，首先感謝在這段製作歷程之中一路上指導並給予我們支持的教授陳詩豐老師，從一開始將我們帶領至完全不熟悉的機電整合領域，到整個專題的製作完成，老師都不斷地接納我們所提出的問題並解開我們的疑惑，才使得本專題得以順利完成。

另外要感謝是再製作過程中給予我們引導的學長，以及無時無刻都在鼓勵我們的父母，和提供物資支援的普特企業有限公司，沒有了他們的支援，我們將陷入迷茫與沮喪之中。

以及感謝專題製作小組內的各位同仁，能夠犧牲課餘休閒時間前來參與研究與討論、互相勉勵扶持，而在最為關鍵的時刻總能同舟共濟、戰勝困難，是本次專題成功的關鍵。

最後再一次由衷地感謝所有參與專題製作的人員，若沒有了他們，那麼本專題將永遠無法如期完成，因此再次獻上最誠摯的謝意！

目錄

第一章 研究動機與目標	1
1.1 專題研究動機	1
1.2 目標流程	2
第二章 材料介紹	3
2.1 硬體	3
2.2 軟體	16
第三章 軟體安裝與程式設計	19
3.1 樹莓派環境中的軟體安裝	19
3.2 OpenCV 影像處理及應用	22
3.3 機械手臂程式設定	32
3.4 自走車程式設計	35
第四章 設備設置與實作流程	36
4.1 產線地面	36
4.2 樹莓派硬體設置	37
4.3 鏡頭與手臂位置設置	38
4.4 自走車安裝與設計	39
第五章 實驗結果與展示	45
5.1 產線完成圖	45
第六章 未來展望	46
第七章 結論	47
參考文獻	48
附錄 A	49
附錄 B	59
附錄 C	60
附錄 D	68
附錄 E	69

圖與表目錄

圖 1-1 產線系統流程圖	2
圖 2-1 各式電子器材	3
圖 2-2 裝在 Arduino uno 上的 Adafruit Proto Shield 擴張板	4
圖 2-3 L298N 步進馬達控制模組	5
圖 2-4 裝在車頭底部的 2 顆 TCRT5000 紅外線感測器	6
圖 2-5 裝在車頭上的 HC-SR04 超音波感測器	7
圖 2-6 組成柵欄的 SG90 伺服馬達	8
圖 2-7 裝在車底提供動力的 DC3V-6V 直流減速馬達	9
圖 2-8 從左至右分別為 9V、3V、6V 電池盒	10
圖 2-9 Raspberrypi 3 Model B+	11
圖 2-10 於產線中架高的 C310 HD 網路攝影鏡頭	12
圖 2-11 Uarm Swift Pro 多功能機械手臂	13
圖 2-12 Uarm Swift Pro 手臂有效載重負荷區域圖	15
圖 2-13 Uarm Swift Pro 手臂細節尺寸圖	15
圖 2-14 Arduino Software IDE 的編寫介面	16
圖 2-15 Raspbian 的主桌面環境	17
圖 2-16 Spyder 的編寫介面	18
圖 3-1 在 Win10 環境下運作的 Anaconda	20
圖 3-2 時不時出現的 404 錯誤	21
圖 3-3 原理示意圖	23
圖 3-4 成果圖，在嚴重的陰影干擾下也能取得正確的結果	23
圖 3-5 常見的二值化處理種類	24
圖 3-6 大津二值化示意圖	25
圖 3-7 高斯模糊運行示意圖	26

圖 3-8	空間矩公式.....	29
圖 3-9	力矩公式.....	29
圖 3-10	SDK 包內測試手臂基本移動的程式.....	32
圖 3-11	裝貨點手臂的佈局示意圖.....	34
圖 3-12	自走車程式流程圖	35
圖 4-1	產線軌道圖.....	36
圖 4-2	樹莓派總體接線圖	37
圖 4-3	塑膠白色瓦楞板所製成的簡易固定器.....	38
圖 4-4	尚未進行改裝的 mbot 機器人.....	39
圖 4-5	自走車佈局圖	40
圖 4-6	自走車物架圖	41
圖 4-7	最初的循線方案	43
圖 4-8	第二種循線方案	43
圖 4-9	自走車接線示意圖	44
圖 5-1	產線系統完成圖	45
圖 5-2	產線系統完成上視圖	45
表 1	Uarm Swift Pro 多功能機械手臂規格表	15

第一章 研究動機與目標

1.1 專題研究動機：

在一次的課堂之中我們看見了來自中國大陸的學生所做的專題報告，因而受到了他們的啟發，使我們窺見了機械視覺強大的功能性，而我們也對程式語言部分深感興趣，有鑑於此，我們希望能透過專題研究與實驗的機會，對程式語言這方面有基本的認識，並研究如何利用程式語言來控制機械手臂及自走車，達成學習自動化機電整合系統的理想。

1.1.1 功能與特色：

使用機械手臂來夾取物體，將物體放置在自走車上，使用 Arduino 來控制自走車，抵達上下貨區時停止自走車等待貨物上貨及下貨，結束上下貨後繼續走到下一個停止區繼續執行上下貨。此專題之產線系統最大特色為可以大量代替人力運輸，節省成本及減少因搬運而產生的事故並提高生產的效率，以創造工廠的競爭優勢。

1.2 目標流程：

本次專題產線系統目標預計為將貨料區堆放的貨物經由機械視覺判定後夾取至自走運輸車之上，再運輸到指定的卸貨點，並經由第二台手臂進行卸貨與分類。其流程圖如下所示：

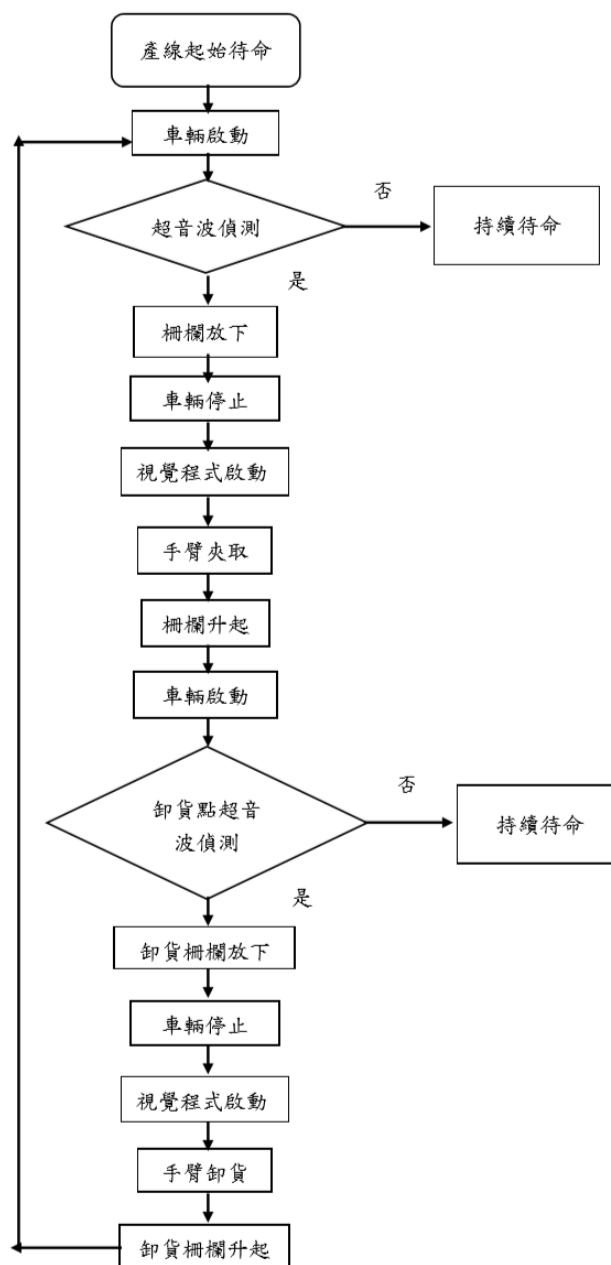


圖 1-1 產線系統流程圖

第二章 材料介紹

本次專題中所運用到的資材介紹：

2.1 硬體：

2.1.1 各式電子器材：

本次專題製作運用到許多機電設備，故需使用到許多電子材料（如下圖 2-1 所示）因杜邦線接頭擁有不須焊接之優點，所以我們使用了該線材來讓各式設備互相通訊及傳遞電能。而麵包板正是連接各個杜邦線最為理想的運作平台，在此次專題製作中考量到了產線緊密狹小的空間，故我們選用了較小規格（ $8.3 \times 5.5 \text{ cm}$ 400 孔）的麵包板。此外我們也使用了可變電阻（即電位器）來控制電壓的大小，進而更精確的來操控各個組件。

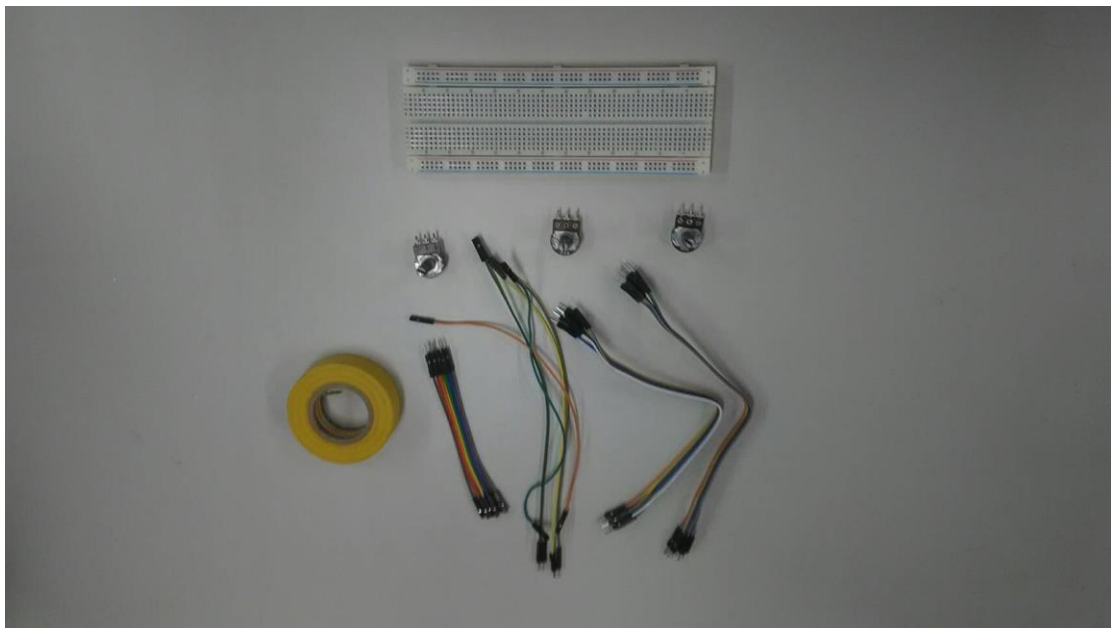


圖 2-1 各式電子器材

2.1.2 Arduino uno 邏輯電路板：

本次專題製作中我們選用了體積小容易控制的 Arduino uno 開發板來做為無人搬運車的主體，在製作的過程中我們發現 Arduino 主板的腳位設置與我們自走車的布線及組件格局不合，因此我們決定使用 Adafruit Proto Shield 擴張板來解決車輛配置的問題(如圖 2-2 所示)。同時我們也運用了 Arduino 的類比與數位腳位來處理感測器的訊號。Arduino 的強大擴展性也讓我們能夠使用許多的擴張模組，已完善我們的目標。

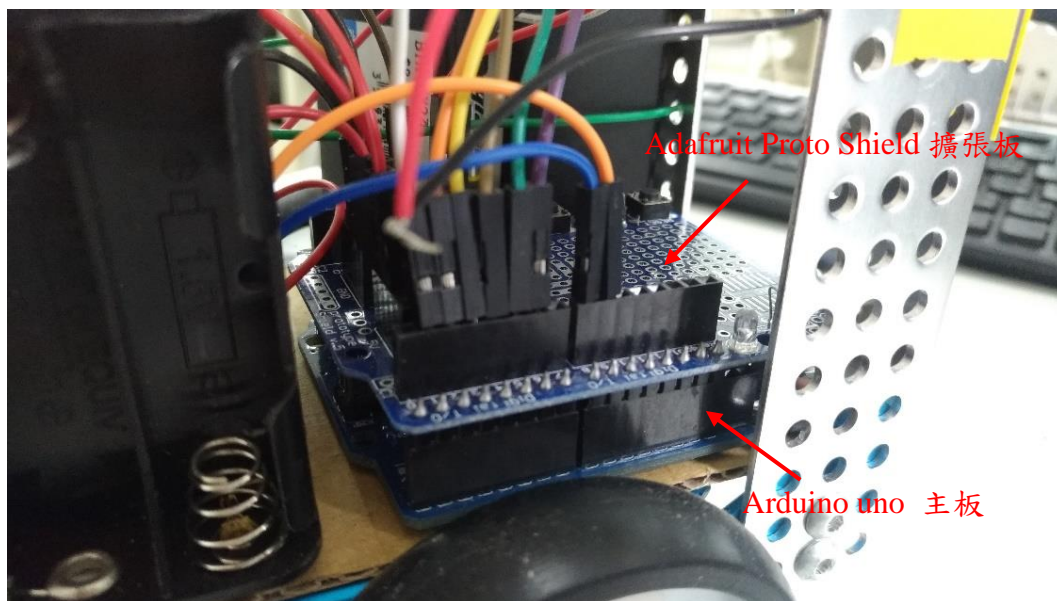


圖 2-2 裝在 Arduino uno 上的 Adafruit Proto Shield 擴張板

2.1.3 L298N 步進馬達控制模組：

本專題的減速馬達使用 L298N 步進馬達控制模組（如圖 2-3 所示）來控制，我們使用總共 12V 的電來使 L298N 驅動兩顆減速馬達，以讓兩顆馬達的動能輸出正常，在組建過程中，我們發現 L298N 的 ENA 及 ENB 的腳位能利用數位訊號的大小來精確控制馬達的轉速，當訊號越大時馬達的轉速越快，訊號越小時速度則越慢，由此能夠藉由程式來將馬達調整至我們所需要的速度。

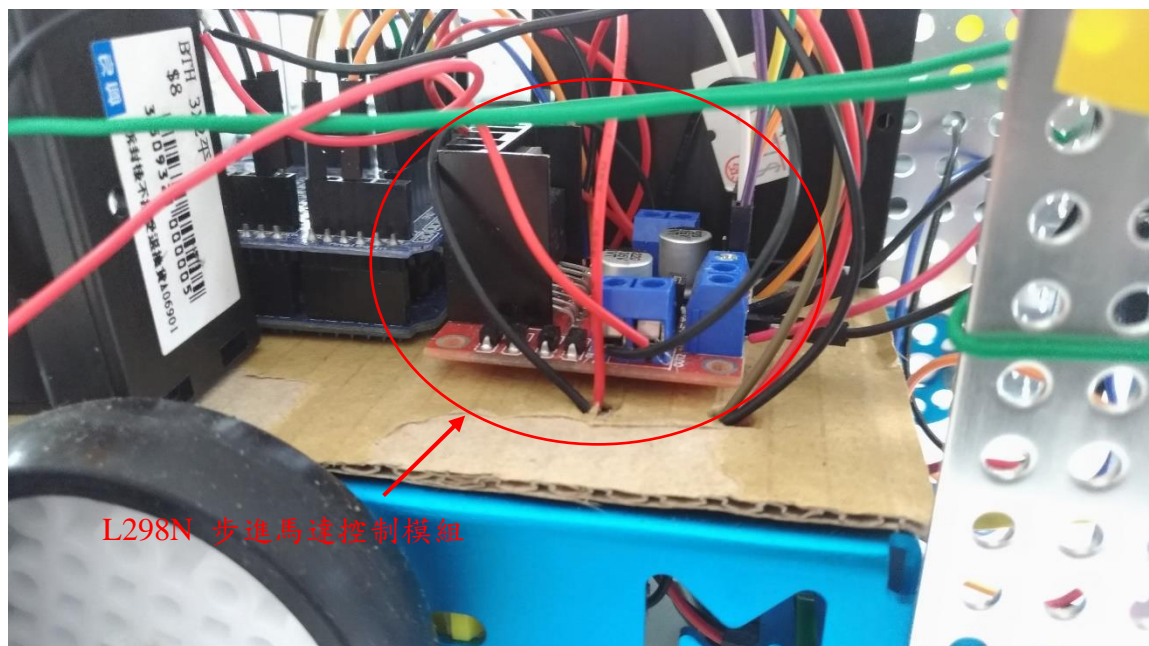


圖 2-3 L298N 步進馬達控制模組

2.1.4 TCRT5000 紅外線感測器模組：

我們認為 TCRT5000 紅外線感測器模組（如圖 2-4 所示）是循線自走車判定路線的最佳選擇，此型號的紅外線循線模組不僅具有訊號乾淨、易於使用等優點，且也更符合經濟效益。感測器板上也具有小型的電位器讓我們能夠調整紅外線感測器的靈敏度，且感測器所輸出的類比訊號也具有使用數值控制之優點，能夠更精確地分辨出產線地面上的黑線，而不會被其他的雜色干擾，所以我們決定使用感測器的類比插腳來傳輸類比訊號至 Arduino 主板。



圖 2-4 裝在車頭底部的 2 顆 TCRT5000 紅外線感測器

2.1.5 HC-SR04 超音波感測器模組：

超音波感測器模組 HC-SR04（如圖 2-5 所示），算是常見的感測器模組，可用來測量前方障礙物的距離。它的運作原理很簡單，此感測器會送出 8 個 40khz 的聲波，如果前方有障礙物，聲波就會返回，感測器收到聲波後，再利用返回的時間，來計算聲波碰觸障礙物之間的距離。該感測器的價格便宜，易於使用，在本次專題中從自走搬運車的車頭到產線的近接偵測區都可以見到此感測器活躍的身影。

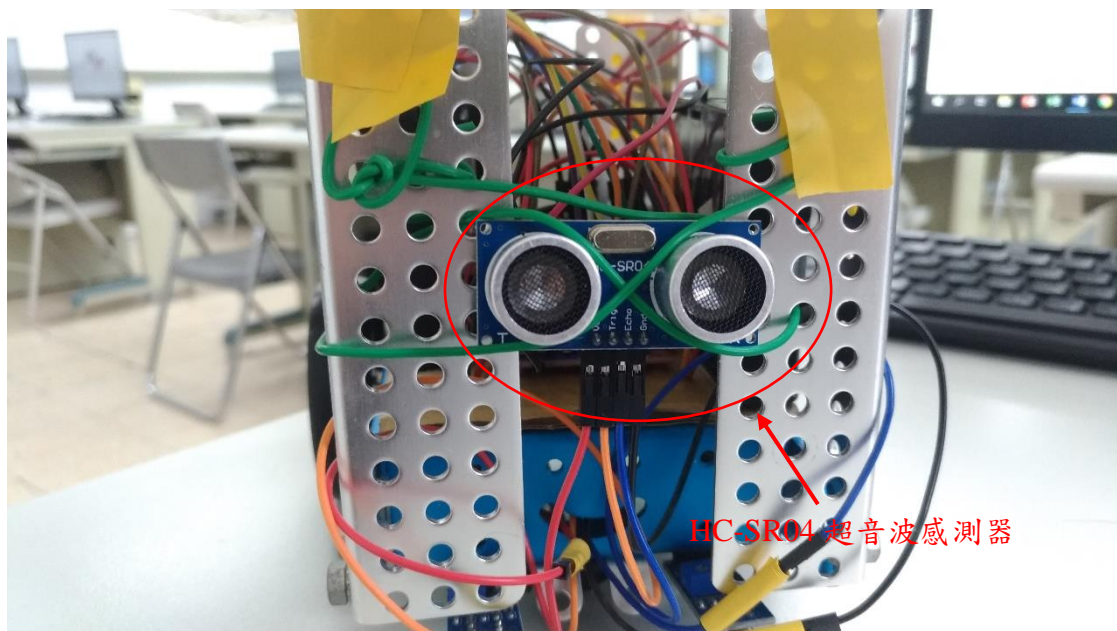


圖 2-5 裝在車頭上的 HC-SR04 超音波感測器

2.1.6 SG90 伺服馬達：

伺服馬達具有返回自身位置，而其中的閉迴路系統也給予了馬達極大的控制精確性，並且擁有迅速的響應速度，本專題中無人運載車在進入手臂工作區時需要類似於停車場柵欄之機構系統來使車子能夠準確地停靠在目標地點。而 SG90 伺服馬達（如圖 2-6 所示）正符合本次專題控制柵欄的理想需求。而步進馬達則是因為響應速度較慢、採用步進點控制較為不便因而沒有列入考慮中。

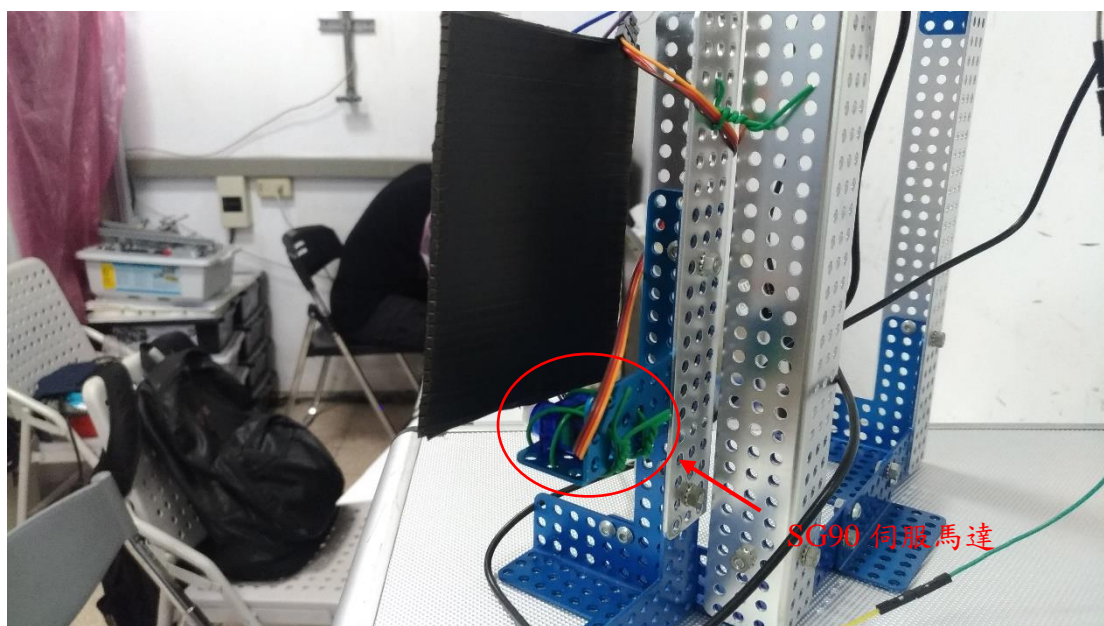


圖 2-6 組成柵欄的 SG90 伺服馬達

2.1.7 DC3V-6V 直流減速馬達：

此減速馬達（如圖 2-7 所示）具有價格便宜、轉速快等優點，且可配合 L298N 步進馬達控制模組來使用，因此成為了本專題自走車的動力來源首選。但缺點是控制精確性並不高，因此導致有時自走車會因兩輪轉速不一致而造成車輛前進時一邊速度過快之情形產生，需要利用程式來控制 L298N 來輸出 PWM 訊號使兩顆馬達轉速能夠盡量達成一致。

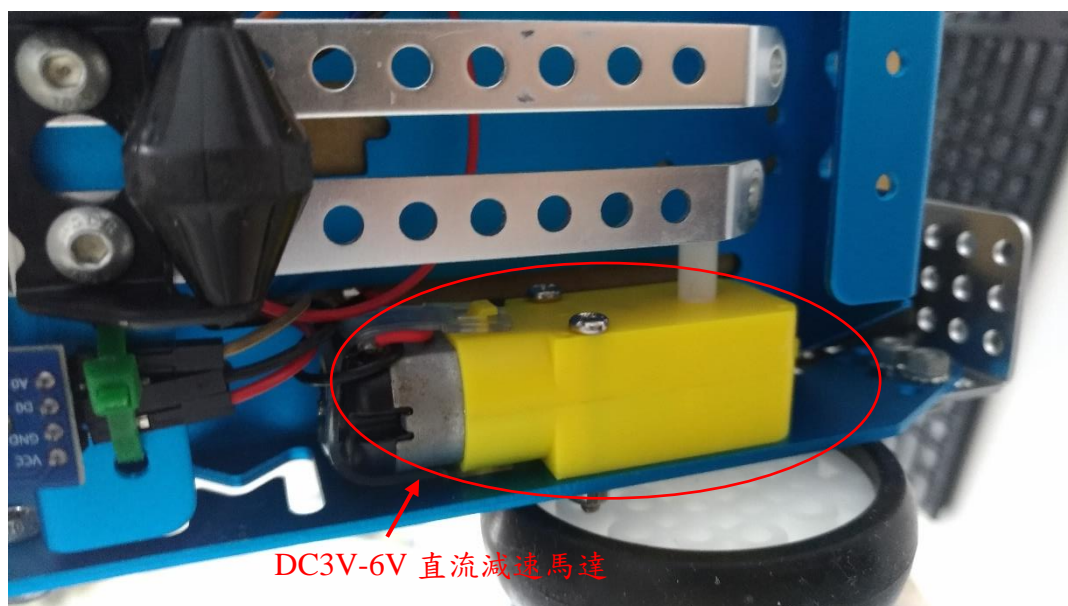


圖 2-7 裝在車底提供動力的 DC3V-6V 直流減速馬達

2.1.8 電池盒：

本次專題所使用的三種電池盒（如圖 2-8 所示）當中，6V 電池盒主要串接兩個為一組來使電壓達到 12V，並且連接至 L298N 的電源輸入孔來驅動無人搬運車的兩顆減速馬達。而 Arduino uno 主板供電的部份，起初我們只使用單顆 9V 的電池來連接主板預設的 DC 2.1 插座，然而我們很快地就發現了 Arduino uno 工作不穩定的問題，爾後判定不穩定的原因應為主板 DC 2.1 插座要求工作額定電壓為 9V~12V，因此我們在此多串接了一個 3V 電池盒，並改為使用主板的 Vin 電源供應腳位，並順利的解決了此問題。

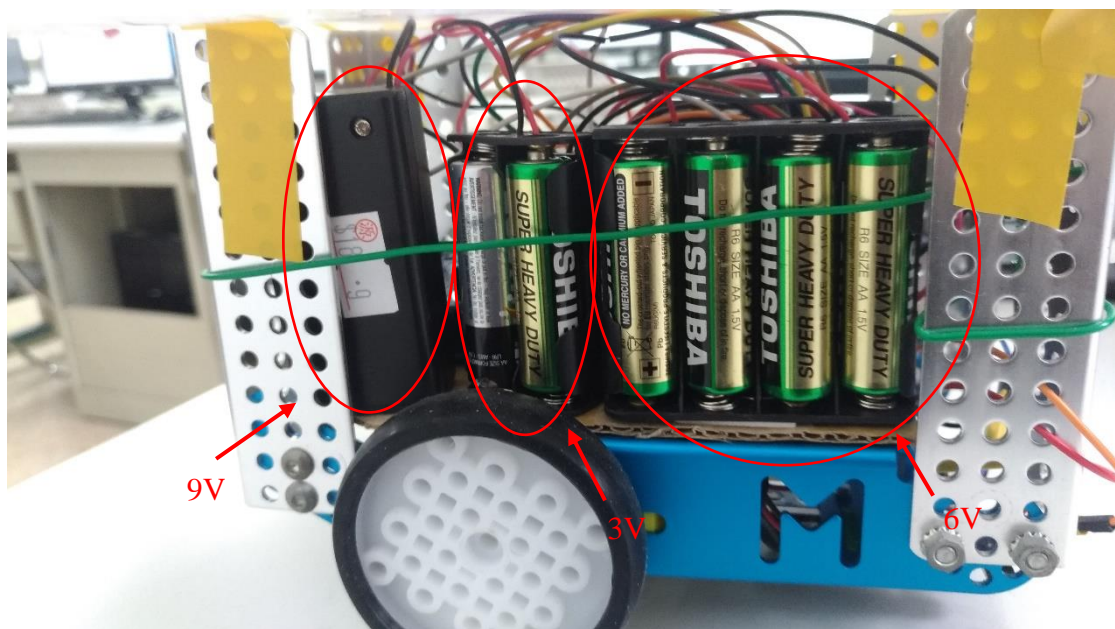


圖 2-8 從左至右分別為 9V、3V、6V 電池盒

2.1.9 Raspberry Pi 3 Model B+（以下簡稱樹莓派）：

樹莓派（如圖 2-9 所示）是價格便宜的微型電腦，由 Raspberry Pi 非營利基金會在 2012 年推出，可搭載開源的 Linux 作業系統，在本專題中，嬌小的樹莓派在緊湊而狹小的產線裡扮演了舉足輕重的角色。在整個產線系統中樹莓派主要負責控制攝影機鏡頭以及機械手臂，並也連接了伺服馬達和超音波感測器來控制自走車的停靠，可以說是本次專題中產線系統的大腦。



圖 2-9 Raspberrypi 3 Model B+

2.1.10 羅技 C310 HD 網路攝影鏡頭：

此網路攝影機（如圖 2-10 所示）具有 720P 高畫質、方便調整網路攝影機鏡頭的角度、體積小等優點，在本專題中主要用來當作機械視覺的影像接收與捕捉器，並藉此利用程式來判別物體。攝影機鏡頭本身獲取到的影像資料會因環境、設置的參數等因素而影響到影像畫面的品質，必須要使用程式更改參數來進行修正。攝影機鏡頭的 USB 接頭則是直接連接至樹莓派 USB 連接埠中，並讓樹莓派擔任影像處理器的角色。



圖 2-10 於產線中架高的 C310 HD 網路攝影鏡頭

2.1.11 Uarm Swift Pro 多功能機械手臂：

在現代的生產線中，繁瑣的傳遞物品和上、下料動作已逐漸被自動化機械所取代，其中機械手臂為執行此類工作的最佳選擇。在本次專題中，深圳市眾為創造科技公司所出品的機械手臂（如圖 2-11 所示）就擔任如此的任務。其中我們為手臂加入了機械視覺判別，只要攝影機鏡頭照到工作區域內的物品，機械手臂就會針對物品的座標進行夾取，如此一來就省去了繁瑣的座標計算及輸入工作，並達到自動化的效果。



圖 2-11 Uarm Swift Pro 多功能機械手臂

表 1 Uarm Swift Pro 多功能機械手臂規格表[1]

基本參數		
	uArm Swift	uArm Swift Pro
重量	1.2kg	2.2kg
自由度	4	4
精度	5mm	0.2mm
負載	500g	500g
臂展	50mm~320mm	50mm~320mm
最大末端運動速度	167mm/s	100mm/s
連接方式	Micro USB	Micro USB
無線連接	Bluetooth 4.0	Bluetooth 4.0
工作電壓	DC 6V	DC 12V
電源適配器	Input:100~240V 50/60Hz Output: 6V5A 30W	Input:100~240V 50/60Hz Output: 12V5A 60W
額定功率	30W	60W
工作環境溫度	0~40°C	0~40°C
硬體參數		
驅動方式	舵機	步進電機+減速器
回饋機制	電位器	12 位元磁編碼器
減速機	舵機齒輪箱	自主研發減速機
機身尺寸 (長*寬*高)	150mm*132mm*281mm	150mm*140mm*281mm
控制主機板	Arduino MEGA 2560	Arduino MEGA 2560
機身材料	鋁合金	鋁合金
串列傳輸速率	115200 bps	115200 bps
可拓展 I/O 介面	I/O *3 , IIC *1	I/O *27 , IIC *1 , 5V*1 , 12V*1 , Stepper*1
軟體參數		
PC 控制軟體	uArm Studio	uArm Studio
移動端軟體	uArm Play	uArm Play
可支援開發環境	Python/Arduino/ROS	Python/Arduino/ROS
開源特性	開源	開源

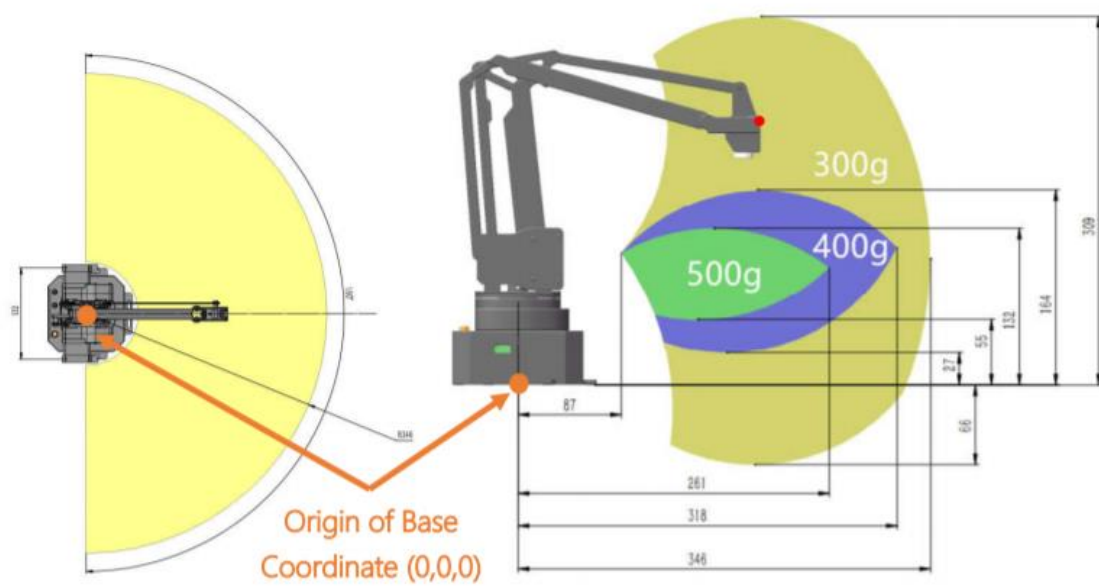


圖 2-12 Uarm Swift Pro 手臂有效載重負荷區域圖




圖 2-13 Uarm Swift Pro 手臂細節尺寸圖

2.2 軟體：

2.2.1 Arduino Software IDE：

本次專題製作之中，Arduino uno 控制板需要使用 C 語言來進行編寫與控制，而我們選擇了相較於我們較為熟悉的 C++來對 Arduino 板進行編程。在此部分 Arduino 官方所提供的免費軟體 Arduino Software IDE（如圖 2-14 所示）本身具有自動縮排、括號匹配、語法突顯等功能，對於程式編寫學習者來說相當的友好，正為我們本次專題理想的 C++程式編寫環境。



```
POT_move

const int In1 = 1;
const int In2 = 2;
const int In3 = 3;
const int In4 = 4;

const int sensor_left= A2; //左循環感測器宣告
const int sensor_amid= A1; //中循環感測器宣告
const int sensor_right= A0; //右循環感測器宣告

int SLL; //左循環感測器狀態宣告
int SAA; //中循環感測器狀態宣告
int SRR; //右循環感測器狀態宣告
int ENA = 6; //左馬達
int ENB = 5; //右馬達

void setup() {
  Serial.begin(9600);
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(sensor_left, INPUT);
  pinMode(sensor_amid, INPUT);
  pinMode(sensor_right, INPUT);
}
```

圖 2-14 Arduino Software IDE 的編寫介面

2.2.2 Raspbian：

在使用樹莓派前，我們首先需要確定樹莓派電腦的用途，以便在之後能夠為樹莓派選擇一套合適的作業系統。在本次專題中樹莓派在產線系統裡負責了許多的任務，因此我們需要在樹莓派內進行編程，且需要一定的穩定性，同時記憶體及 CPU 空間佔用量也不能過大，在此過程中我們考慮過使用 Windows 10 Iot Core、Ubuntu 和 Raspbian，其中我們認為 Windows 10 Iot Core 和 Ubuntu 多了許多不必要的功能，比較不適合此次的需求，因此最後我們選擇了能夠在樹莓派上穩定運行的 Raspbian（如圖 2-15 所示）。



圖 2-15 Raspbian 的主桌面環境

2.2.3 Spyder :

在決定要如何利用程式來進行影像處理以及控制機械手臂時，我們選擇了簡便、擴展性與通用性強大的 Python 語言，而我們認為針對 Python 語言來說最好的集成開發環境莫過於 Spyder（如圖 2-16 所示）。Spyder 為輕量級的 IDE，因此所佔用的系統資源較少，很適合使用樹莓派的情況，同時 Spyder 也有類似於 Matlab 的圖像顯示窗口，非常適合讓我們用來查看與比對影像處理的結果。

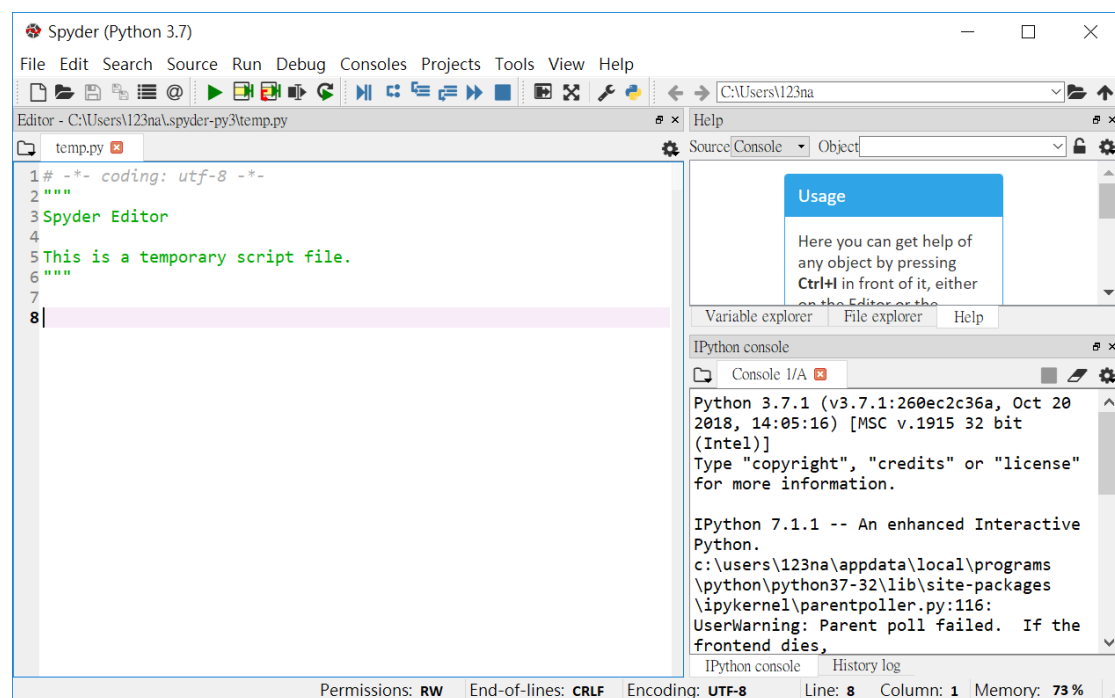


圖 2-16 Spyder 的編寫介面

第三章 軟體安裝與程式設計

本次專題產線系統中所使用的機械視覺功能，乃是利用 Python 語言來進行編寫，而所有視覺程式當中最為重要的就是引入 Intel 公司免費釋出的開源電腦視覺函式庫（Open Source Computer Vision Library，以下簡稱 OpenCV），此函式庫擁有解決臉部辨識、手勢辨識、物體辨識、人機互動、運動跟蹤等問題的能力，且支援多種程式語言，為當今熱門的開源機械視覺函式套件。

3.1 樹莓派環境中的軟體安裝：

在前章所提到的 Raspbian 作業系統，是為樹莓派基於 Debian 上量身打造的作業系統，而 Debian 又為類 Unix 作業系統，也因此本次專題中所有在樹莓派平台上安裝的各式套件，以及樹莓派系統本身的設置，都須使用類 Unix 作業系統的指令來完成。

3.1.1 Python 版本更新：

在樹莓派上使用 Python 語言編程前，需要事先將 Python 更新至 3.6 以上之版本，以免在編寫程式時遭遇到版本之間不相容的錯誤。在此部分的安裝之中使用了 conda 軟體包管理系統來進行版本的下載與安裝，而一般桌上型 Windows 電腦上所安裝的 conda 系統，即為眾所皆知的 Anaconda（如圖 3-1 所示），但對於樹莓派來說，Anaconda 安裝時附帶了太多非必要的套件與圖形介面，這些累贅無疑大幅地增加了樹莓派的負擔，甚至影響到了其運作，因此在此部分中我們使用了 Anaconda 的刪減版，「Miniconda」，一個輕便且與樹莓派相容性更好的版本（詳細安裝指令見附錄 B）。

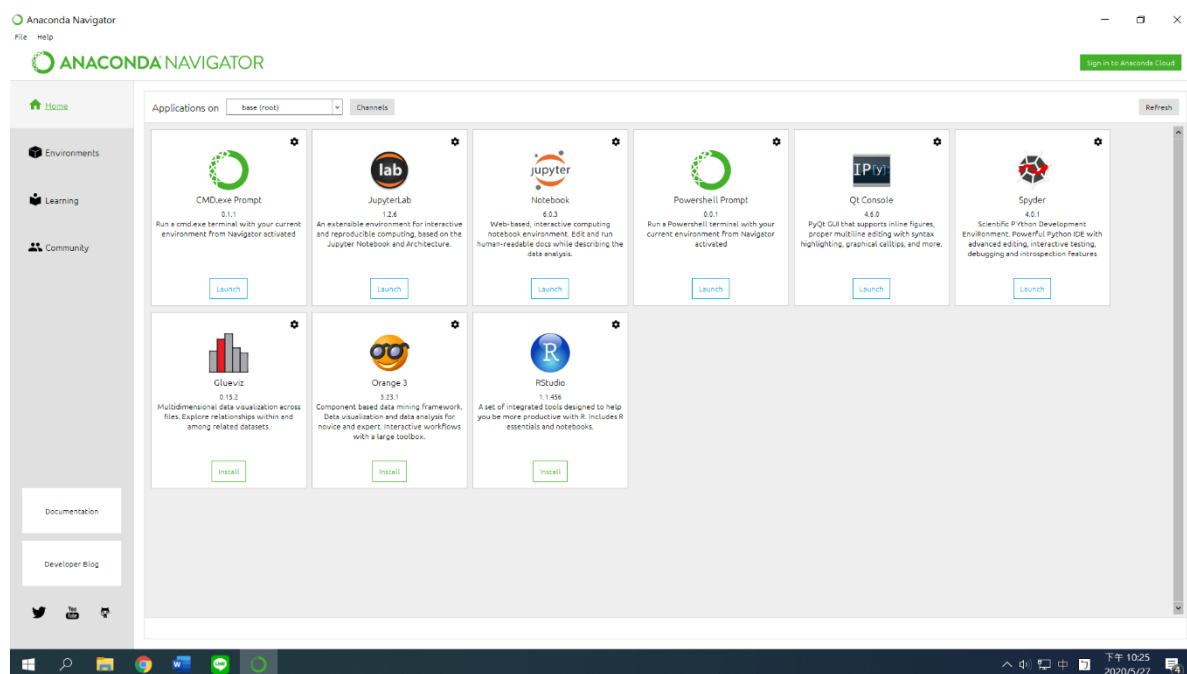
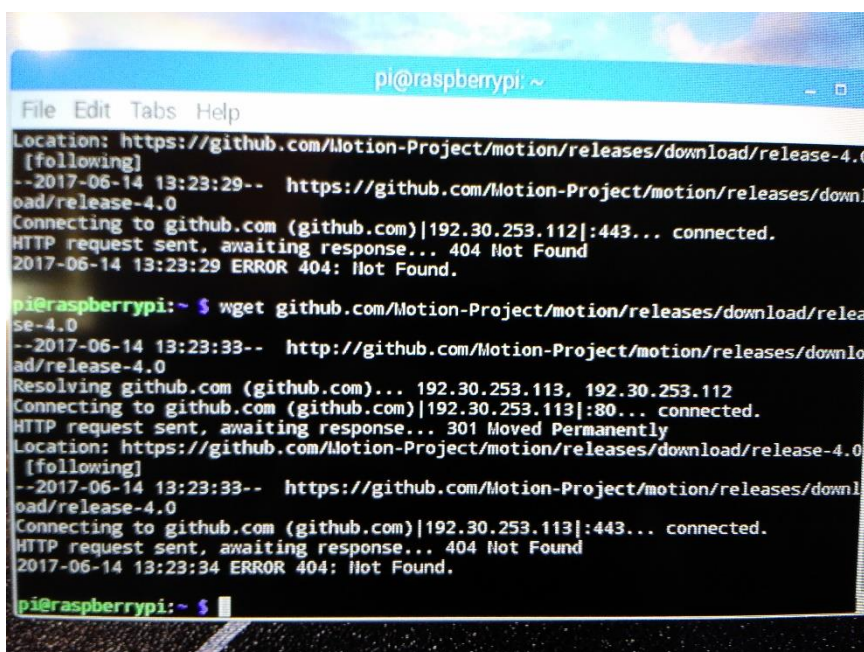


圖 3-1 在 Win10 環境下運作的 Anaconda

3.1.2 Spyder 與 OpenCV 的安裝：

在此部分的安裝中，我們遭遇到了許多安裝上的問題，這些問題大部份都出自於指令所呼叫的網路安裝來源中，因套件版本過時、安裝源過久無人維護、或者只是單純網路上的問題，這些因素都導致了安裝上的諸多不便（如圖 3-2 所示），也因此我們時常需要交替使用不同安裝的方式如類 Unix 作業系統的 `sudo apt-get` 指令、Python 內建的軟體包管理系統 `pip`、以及前文所提到的 `conda` 系統來克服這些障礙。而 OpenCV 的安裝部份則是需要打開 Spyder IDE 並使用 Python 指令 `import cv2` 來檢查 Spyder 所缺少的其他 OpenCV 必要套件並一一進行安裝（詳細安裝指令見附錄 B）。

A terminal window on a Raspberry Pi showing two failed wget attempts. The first attempt to download from a GitHub release page fails with a 404 error. The second attempt, after changing the URL to a standard HTTP path, fails with a 301 redirect error, which then leads to another 404 error when following the redirect.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
Location: https://github.com/Motion-Project/motion/releases/download/release-4.0  
[following]  
--2017-06-14 13:23:29-- https://github.com/Motion-Project/motion/releases/downl  
oad/release-4.0  
Connecting to github.com (github.com)|192.30.253.112|:443... connected.  
HTTP request sent, awaiting response... 404 Not Found  
2017-06-14 13:23:29 ERROR 404: Not Found.  
  
pi@raspberrypi:~$ wget github.com/Motion-Project/motion/releases/download/relea  
se-4.0  
--2017-06-14 13:23:33-- http://github.com/Motion-Project/motion/releases/downlo  
ad/release-4.0  
Resolving github.com (github.com)... 192.30.253.113, 192.30.253.112  
Connecting to github.com (github.com)|192.30.253.113|:80... connected.  
HTTP request sent, awaiting response... 301 Moved Permanently  
Location: https://github.com/Motion-Project/motion/releases/download/release-4.0  
[following]  
--2017-06-14 13:23:33-- https://github.com/Motion-Project/motion/releases/downl  
oad/release-4.0  
Connecting to github.com (github.com)|192.30.253.113|:443... connected.  
HTTP request sent, awaiting response... 404 Not Found  
2017-06-14 13:23:34 ERROR 404: Not Found.  
  
pi@raspberrypi:~$
```

圖 3-2 時不時出現的 404 錯誤

3.2 OpenCV 影像處理及應用：

3.2.1 影像處理：

在機械視覺中，我們需要讓電腦即時讀取一幀一幀的影像，並讓電腦能夠藉著讀取到的影像來進行識別。然而讀取到的影像可能擁有品質不佳、色調或光影混亂、雜訊過多、陰影遮蔽等不確定因素存在，這些因素都會使電腦在辨別的過程中產生或多或少的誤差，況且若是不做任何的處理就加以辨識的話，會加重處理器的負擔，影響到處理的時間、效率，對於產線系統來說是極其不樂見的。

而除了改變外在環境以外，我們也能夠對電腦讀取到的影像作預先處理，使電腦辨別能夠更順利與準確。OpenCV 本身就擁有需多有關影像處理的函式，其中有些函式雖然功能強大但卻不適合於我們的情況之中，這些都需要進行反覆的實驗與比對以便挑選出最符合我們所需求的方法。以下將介紹本次專題之中所使用的影像處理方法（詳細影像處理程式見附錄 A）。

(1) cv2.absdiff :

此函式中的“absdiff”直接從字面上分開翻譯就是「絕對值的差異」，主要功能為找出兩個圖片之間的差異性[2]，並也可以藉此拿來將影像中的物品從圖片的背景之中分離出來，其原理為將兩個影像中的元素相減，並把相減後的差值取絕對值，最終的輸出結果就為兩個影像的差異物（如圖 3-3 所示）。如此一來不需要應用到機械學習的範疇就能夠使用 OpenCV 獲得到強大的影像辨識成果（如圖 3-4 所示）。

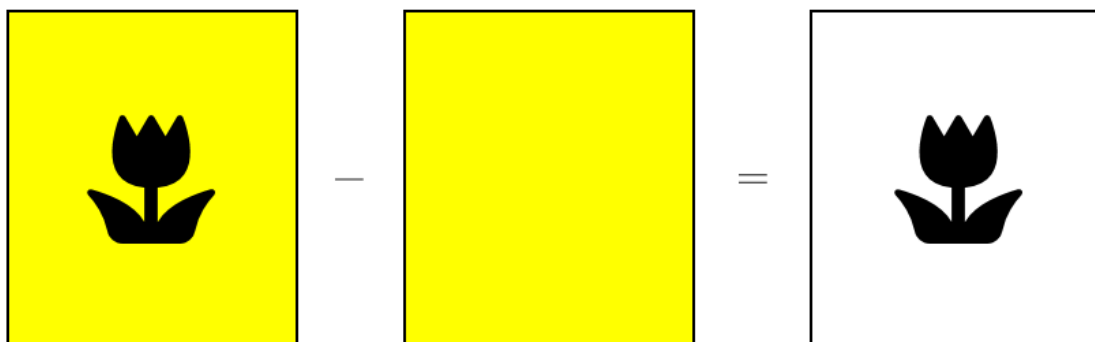


圖 3-3 原理示意圖

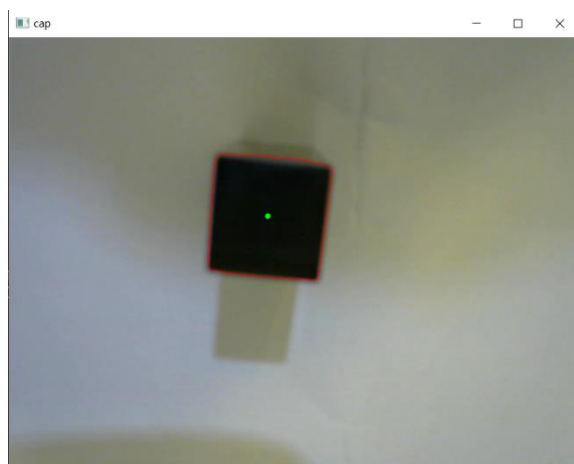


圖 3-4 成果圖，在嚴重的陰影干擾下也能取得正確的結果

(2) 二值化與灰階：

在分離出了物體後，就必須要正式開始對需要辨識的影像進行處理，而影像處理最重要的步驟就是將影像轉為二值化，也就是將影像限制以“黑”與“白”來表現。藉由二值化，就能夠將影像中平常肉眼不易察覺的資訊顯現出來[3]，這些資訊通常為雜訊居多，需要透過後續的處理解決掉。而再進行二值化之前則是需要先將圖片的顏色轉為灰階化，才能夠進行二值化的處理。其原由為將圖片轉為灰階後，二值化會針對圖片中的每個灰階像素點設定一個標準值，稱為“閾值”，將高於或低於此閾值的像素點設為單純的黑色或是白色。二值化處理中有許多針對使用者不同的情況及需求所提供的算法(如圖 3-5 所示)。

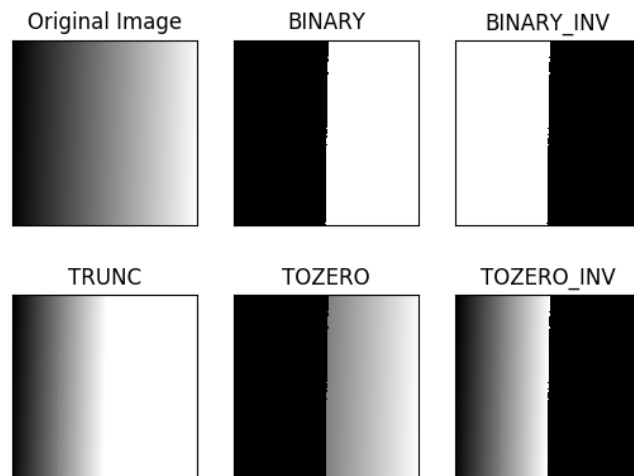


圖 3-5 常見的二值化處理種類

而本次專題中所使用的算法為最為普通的二值化算法，也就是圖 3-5 中的“BINARY”，除此之外為了確定二值化中正確的閾值，以便取得更好的二值化效果，我們又多引入了“大津二值化”（Otsu thresholding），此算法為日本學者大津展之（Nobuyuki Otsu）於 1979 年針對優化圖像二值化議題所提出的高效演算法，此算法能自動為圖像計算出合理的閾值，其原理為將圖像中像素點的分佈列為直方圖（如圖 3-6 所示），並從直方圖中利用公式來找尋適合的閾值[4]。

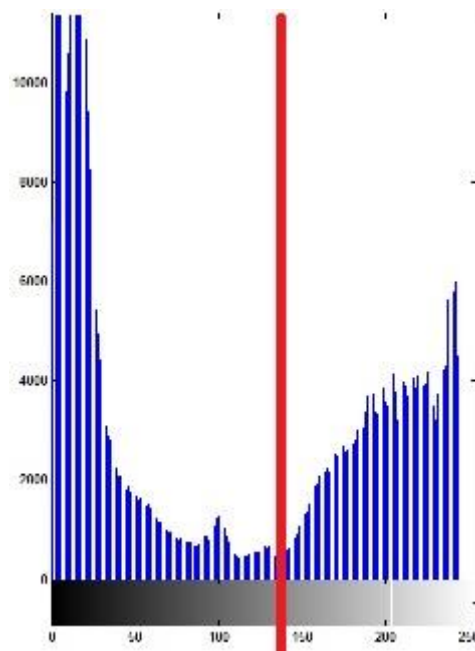


圖 3-6 大津二值化示意圖

(3) 高斯模糊化：

前文提到再二值化處理過後，圖像會產生許多的噪點干擾，這些不必要的干擾通常源自於影像讀取中所產生的、過於精細的像素點，為了處理掉這些噪點，就需要降低影像的精細度，使影像處理系統能夠忽略掉出現的噪點。

而模糊化處理就能夠使圖像變得更加模糊、平滑，進而解決此項問題。在 OpenCV 中就有提供高斯模糊化處理的函式（`cv2.GaussianBlur`），其中函式會計算圖像中的某些像素點與周圍的像素點並取平均值（如圖 3-7 所示），並在整個圖像中進行卷積運算 [5]。

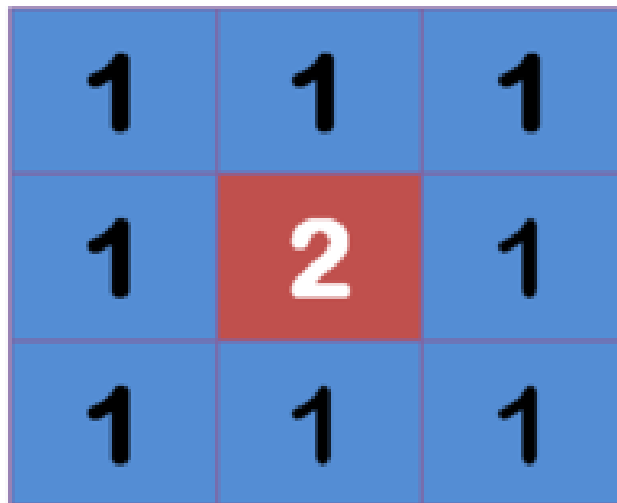


圖 3-7 高斯模糊運行示意圖。在取平均值後，中間的像素點數值會變為 1

3.2.2 輪廓檢測：

圖像模糊化處理後，影像處理器就等於獲得了相較於原圖而言較為純淨的圖像，也就是能夠開始正式對影像中的物體進行辨識。而 OpenCV 當中就擁有許多能夠辨識物體的方法，以下將介紹本次專題中所使用到的兩個函式。

(1) Canny 算法邊緣檢測：

OpenCV 中的 Canny 邊緣檢測函式擁有良好的精確性，在圖像邊緣定位時能夠更好的分離出物體與背景並正確的標定出物體的邊緣。Canny 演算法是由澳洲的計算機科學家約翰·坎尼 (John F. Canny) 於 1986 年所提出，其輪廓檢測運作過程為計算並尋找出每個像素的邊緣梯度和方向，緊接著再檢查並移除其他不相干的像素，最後利用兩個高低閾值來判定邊緣有無接續，若有的話即為真實的邊；若無的話則被捨去（例如在物體外錯誤的邊緣線條）。函式的兩個閾值比例普遍都是設定為 1:2 或 1:3，在偵測上會有較好的效果（本次專題設定為 50 及 150，詳細程式見附錄 A）[6]。

(2) 輪廓檢測：

得出了物體的邊緣後，接下來就需要查找物體的輪廓，以利之後的物體質心計算，而 OpenCV 的 `cv2.findContours` 函式就相當的適合此部分的需求。與前文提到的 Canny 函式不同的是：Canny 函式所尋找的是物體的“邊緣”；而 `findContours` 則是尋找物體的“輪廓”，兩者之間最大的差異為邊緣之間不一定會互相接合，而輪廓則是擁有高度的連續性。

也因此我們在實驗中發現雖然能夠跳過 Canny 邊緣檢測而直接使用 `findContours` 來找尋物體的輪廓，但是輪廓對影像有著相當嚴格的要求，只要稍有偏差、或是物體不工整，程式判定就十分容易出錯。而對照事先使用 Canny 來預先幫 `findContours` 找出邊緣並加以判斷輪廓的程式，成功機率則是大幅的上升，除非影像攝取環境過於惡劣或是設備上的損壞，否則幾乎無任何問題產生。

3.2.3 尋找物體座標：

辨識出了影像內每個物體的輪廓後，就能夠利用 `cv2.moments` 函式來尋找每個輪廓的圖像矩，其函式的原理源自於計算圖像矩的公式（如圖 3-8 所示），此公式能求出輪廓的空間矩（ $m_{00}, m_{10}, m_{01}, m_{20}, m_{11}, m_{02}, m_{30}, m_{21}, m_{12}, m_{03}$ ），而 m_{00} 即為二值圖像輪廓的面積，接著就能使用物理學的力矩公式（如圖 2-9 所示）來計算出物體質心的 (x,y) 座標[7]。

而若是先前的影像因各式原因導致給出的輪廓品質不好的話， m_{00} 的數值就會等於零，此時程式在計算力矩公式時就會出現無法除以零的數學錯誤，需要設條件式來讓程式遭遇此狀況後給出應對方案。

$$m_{ji} = \sum_{x,y} (\text{array}(x,y) \cdot x^j \cdot y^i)$$

圖 3-8 空間矩公式

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

圖 3-9 力矩公式

3.2.4 物體形狀辨識：

在取得物體座標的同時，我們在視覺的程式區塊裡又編寫了一小段用於物體形狀辨識的程式，在本次的產線系統中有三種不同形狀的物體（如圖 2-10 所示）會從裝貨區分別被夾取並藉由無人運輸車運送至卸貨區，此時卸貨區的視覺處理系統就需要額外再辨識物體的形狀，並用手臂對單個貨物進行分類放置。

在此部分中我們使用了兩個函式來判別物體的形狀，其中形狀的辨識相當的容易受到角度或是物體擺放位置的影響，因此產線的佈局以及無人運輸車的停靠點在此部份就顯得格外得重要。然而解決此問題的最佳方法還需要深入至機械學習領域，本次專題因製作時間上的限制故暫不探討之。

(1) 輪廓周長：

眾所皆知不同的形狀之間皆擁有不同的周長，而這正是我們設立條件式判定形狀的因素之一，而 `cv2.arcLength` 函式就能夠用來計算影像中閉合輪廓的周長，然而很快的我們在實驗中就發現了只使用周長所造成的不確定因素過高，只要物體稍微偏離一些角度，其失真的影像投影就會造成錯誤的周長，這項因素也促使了我們尋求更好方式來解決此項需求。

(2) 輪廓近似點：

我們發現了若使用輪廓近似點的方式來設立條件式，那麼成功的機率就會變高許多，我們能使用 `cv2.approxPolyDP` 來藉由前面得出的周長計算出物體的輪廓近似點，輪廓近似點為一個輪廓中由更少的點的組合，因此運用此方法我們能將程式判斷設為若輪廓近似點數量為 3，此形狀就為三角形；若為 4，此形狀就為正方形；若為 15 以下，此形狀就為圓形，若是超過這個標準，則此形狀就無法辨識。

3.3 機械手臂程式設定：

3.3.1 基礎設定：

Uarm Swift Pro 機械手臂本身支援多種程式編譯，在本次專題中視覺處理的部分由 Python 語言來完成，而為了整合手臂與視覺系統，並避免不同設備平台之間的轉換造成的潛在問題，我們決定同樣使用簡單明瞭的 Python 語言來控制機械手臂，在此部分我們使用官方所提供的開源 Python SDK 包（如圖 3-10 所示），裡面包含了 Python 編程機械手臂所需要的各式函式模組，而我們所編寫的程式只需在此 SDK 環境下就可正常運作（詳細機械手臂程式見附錄 A）。

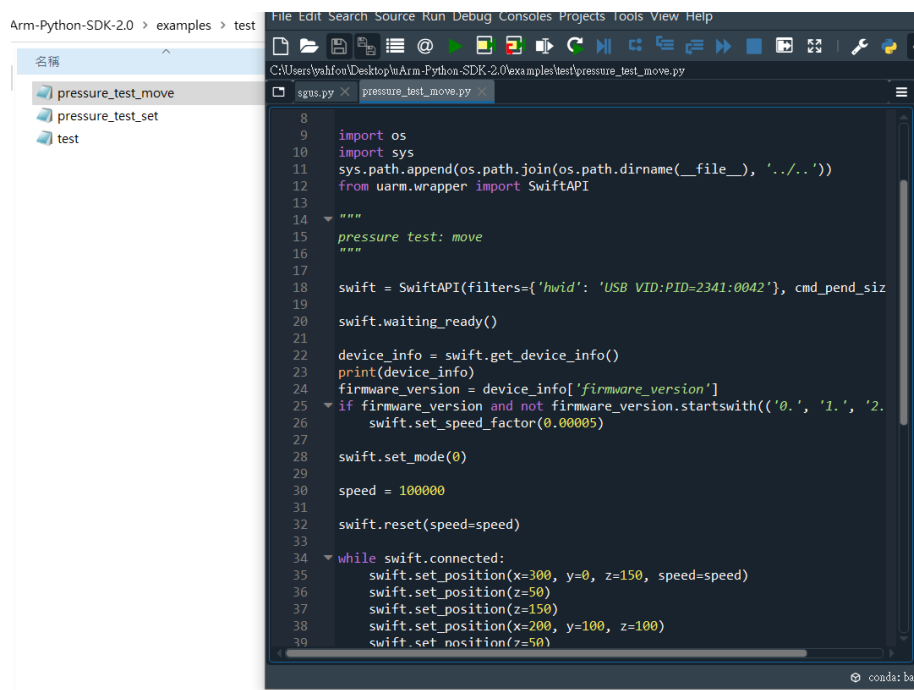


圖 3-10 SDK 包內測試手臂基本移動的程式

3.3.2 手臂與鏡頭座標的轉換：

在前一節中影像尋找物體座標的部分，我們順利取得了影像內所有物體的質心 X、Y 座標，而手臂的部分我們也使用了 SDK 包內所提供的函式來控制手臂的 X、Y、Z 軸，然而這兩座標之間並互不相關，機械手臂的 X、Y、Z 軸為手臂的基座標，而物體質心則是鏡頭內的 X、Y 座標，要如何轉換這兩種座標，我們想到了一種簡單而有效的方法：使用比例來換算。

首先我們使機械手臂移動至鏡頭內原點(位於影像視窗左上角)，並記下該點手臂之座標(假設為 x_1, y_1)，接著我們將手臂移動至鏡頭內某一特定點(假設為 x_2, y_2)，並同時記下該特定點之鏡頭內座標(c_x, c_y)，之後求出 $(x_2 - x_1, y_2 - y_1)$ ，即為手臂原點至鏡頭原點的增量(假設為 x_3, y_3)，緊接著將兩個同方向的 x_3 、 c_x 相除，即可得到 X 方向在兩座標系統間轉換的比例；Y 方向同理，將同方向的 $(y_2 - y_1)$ 與 c_y 相除後即可得到 y 方向之比例(詳細計算過程見附錄 D)。

在程式實際運作時我們將手臂的移動程式座標參數(X、Y)設為：物體質心(cx, cy)座標乘上我們所得出的比例，並直接加上鏡頭原點座標，如此一來就能使手臂移動至實驗物體上，並進行夾取。在此要注意的是，手臂與鏡頭不同的擺放位置與之相對的座標方向也不同（如圖 3-11 所示）。使用此座標轉換方法會有 3mm 左右的誤差距離，但考量到機械夾爪的體積以及物體的大小，並不影響其運作（本次專題實驗物體高度皆相同，Z 值為固定，故而不探討之）。

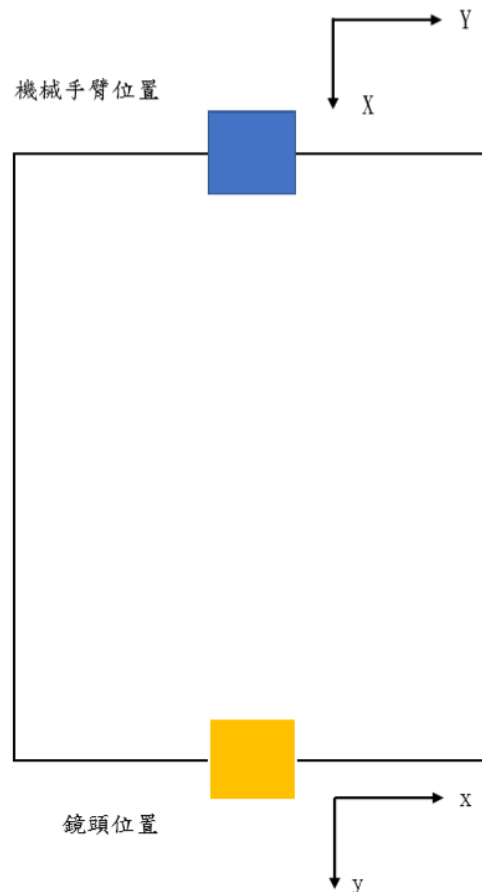


圖 3-11 裝貨點手臂的佈局示意圖，鏡頭 y 方向與之對應的是手臂的 X 方向。

3.4 自走車程式設計：

在此部分中，我們使用 C 語言來控制自走車的行進方式，本次專題中的自走車為滿足多樣的需求，裝設了許多各式的感測器，這些感測器只要 Arduino 一啟動，就會輸出給主板許多的訊號，這些訊號依種類不同也會於 Arduino IDE 中的序列視窗上顯現出各式各樣的數值，而這些數值正是自走車的行進邏輯中重要的條件依據。

其中感測器在不同的地點、不同的時間、不同的設定下所提供的數值都會有所落差，並對此部分中的實驗造成了極多的變因，這些都需要藉由不斷的試驗來發現問題的可能所在，並藉由軟體與硬體之間的調適與配合來使自走車能夠達到最佳的適應狀態，也因此我們將程式分為多個區塊以利於之後的頻繁更動（詳細程式見附錄 C）。其程式基本流程圖如下所示：

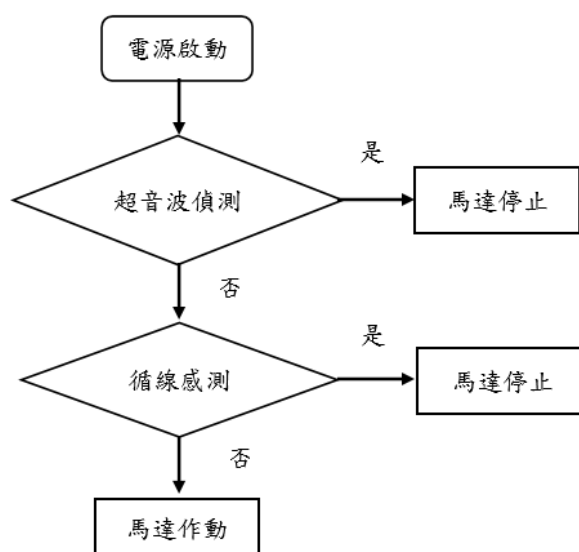


圖 3-12 自走車程式流程圖

第四章 設備設置與實作流程

4.1 產線地面：

在本次的專題產線系統中，我們使用自走車來自動將貨料區的貨物運輸至產線另一端的卸貨點，在此過程中自走車會藉由地面上的黑線來將車輛自動導引至目的地，其中負責此項循跡任務的便是二個 TCRT5000 紅外線感測器。在地面繪製黑線軌跡的當中，我們發現普通黑色麥克筆所繪製的黑線，會使紅外線感測器的反射率過低，導致誤認其黑線顏色為白色，使自走車失控。爾後我們發現黑色的電工膠帶對感測器來說為絕佳的反射體，因此最後我們將電工膠帶貼在白色的塑膠瓦楞板上，使其形成產線系統的路面（如圖 4-1 所示）。

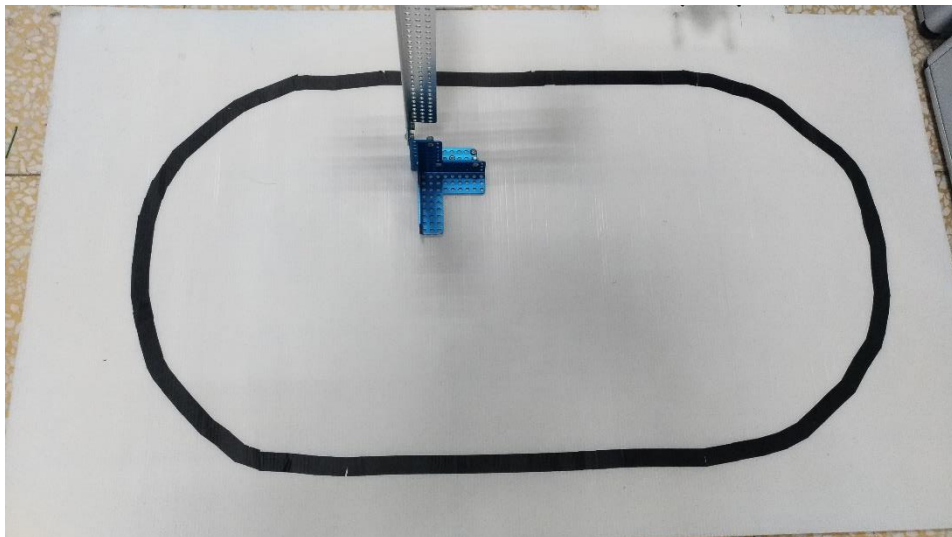


圖 4-1 產線軌道圖

4.2 樹莓派硬體設置：

在產線裡的裝貨區與卸貨區中，樹莓派藉由超音板感測器來偵測逼近的自走車，以此來啟動手臂及鏡頭的主程式，並控制伺服馬達來將柵欄放下。在接線時我們注意到了超音波感測器所輸出的訊號電壓過高（+5V），不適合直接接上樹莓派，因此我們使用分壓器的原理來將電壓分成 3.3V 與 1.7V，在此部份中我們將 B10K 電位器的電阻值轉至 3.3Ω，並將輸出訊號線接至電位器的左側及中間接腳（如圖 4-2 所示，詳細電路圖見附錄 E）[8]。

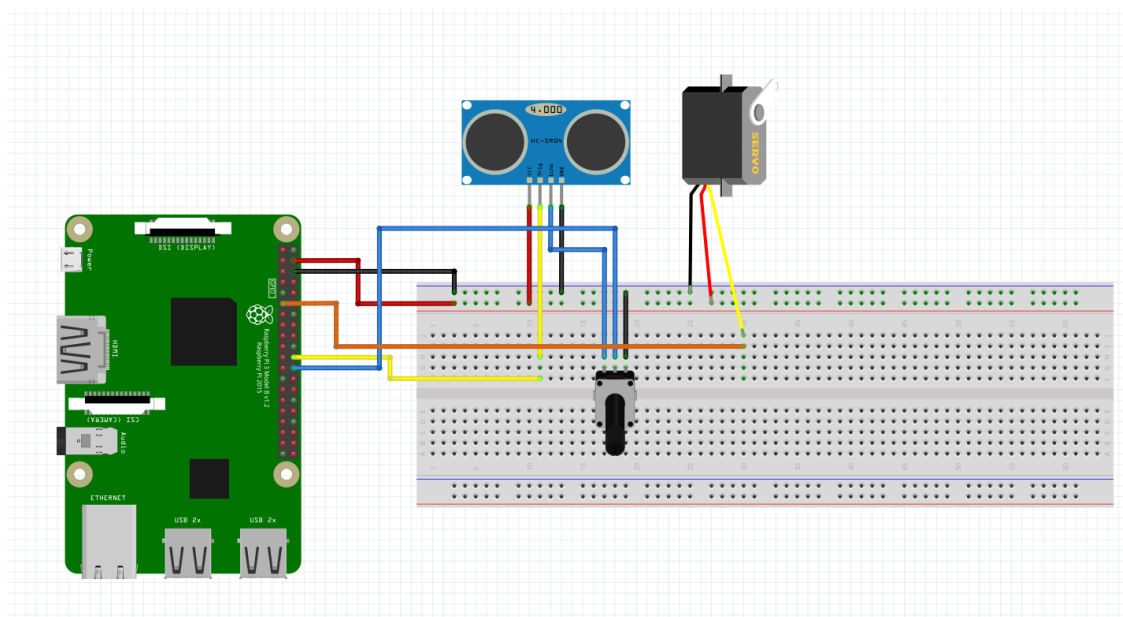


圖 4-2 樹莓派總體接線圖（使用 fritzing 軟體繪製）

4.3 鏡頭與手臂位置設置：

在本產線系統中，因機械視覺與手臂之間的座標定位以及自走車行進位置的關係，鏡頭和手臂往往需要安穩的固定在特定的位置上，因此我們使用多餘的塑膠瓦楞板來割出手臂與鏡頭架的底座形狀（如圖 4-3 所示），並使其之互相契合，再用螺絲與螺帽鎖至地面上的白色塑膠瓦楞板上，如此一來手臂與鏡頭在產線中的相互位置便能良好的固定住。

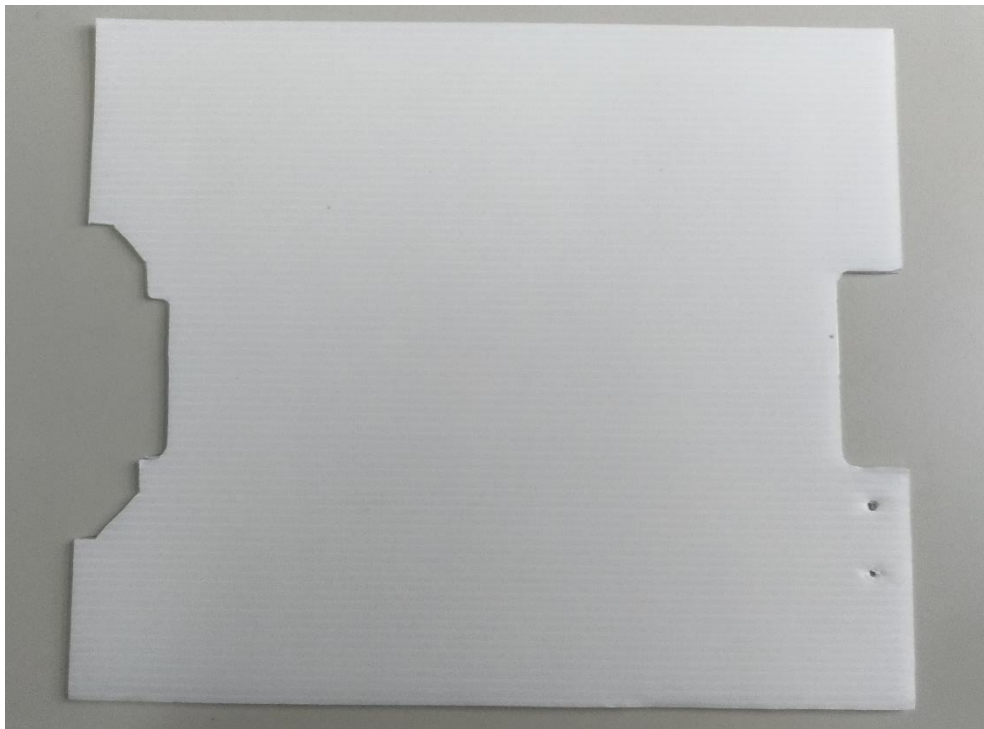


圖 4-3 塑膠白色瓦楞板所製成的簡易固定器

4.4 自走車安裝與設計：

本次專題中的自走運輸車需要滿足產線的多種需求：循線、遇障即停、搭載貨物以及穩定移動，在諸多考量之下，我們認為車架本體的設計與佈局為關鍵因素之一，因此我們決定使用先前比賽得名後所留下來的 mbot 機器人來當作我們的車架底座，mbot 機器人為邁克兄弟科技股份有限公司所生產，其矮小輕便的車形相當地適合讓我們在上面進行延伸與改裝（如圖 4-4 所示），也較為符合經濟效益。

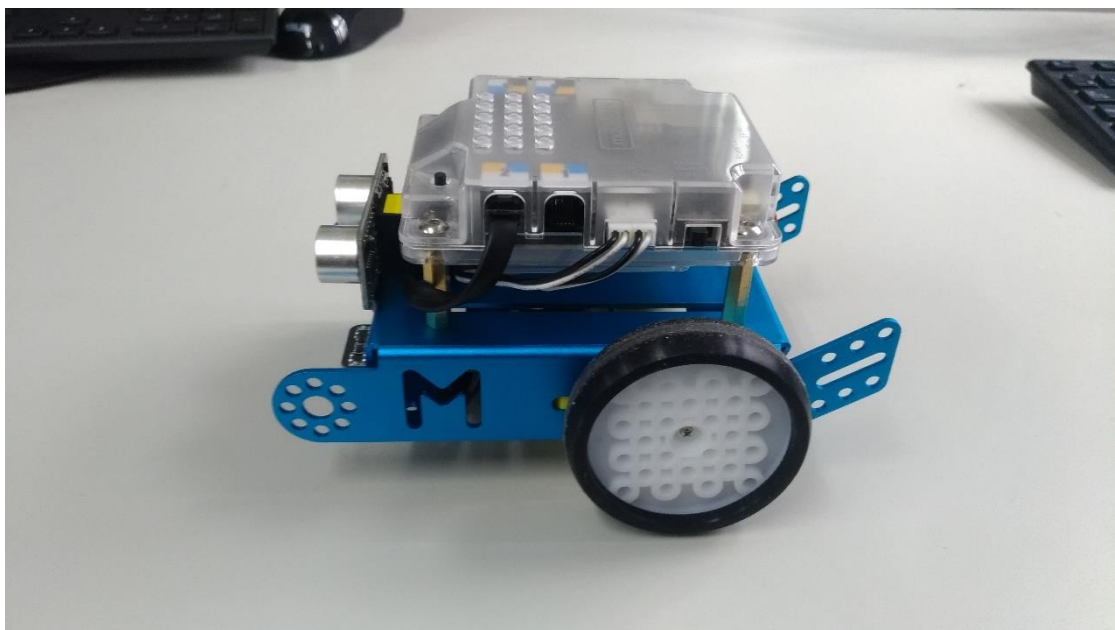


圖 4-4 尚未進行改裝的 mbot 機器人

4.4.1 車身配重：

在不斷的實驗中，我們發現最初的原先設計使得車身配重過重，無法讓自走車的減速馬達轉軸克服其本身的摩擦力，導致自走車動力不足無法行進，因此我們決定在車輛的配置之中進行刪減，將不必要的鋁架更換為厚紙板，並使用泡棉膠將 Arduino 主板、L298N 控制模組以及若干電池盒黏置於車架正中間，使其車體配重集中在中部（如圖 4-5 所示），此舉成功的改良了車體過重的問題，並改善了原先設計配重不均所造成的車體晃動之問題。

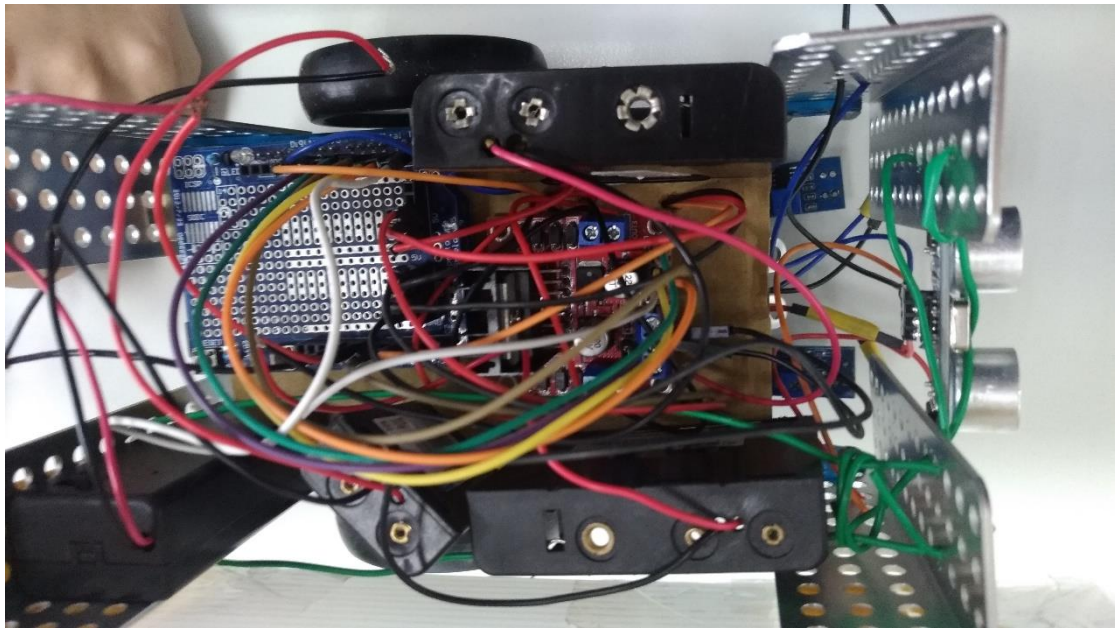


圖 4-5 自走車佈局圖

4.4.2 車體置物架：

在產線系統運作中，裝載貨物的自走車在停靠至卸貨點時，其鏡頭會自動截取自走車物架上的貨物影像，使手臂能夠進行夾取卸貨。因此物架為了配合影像處理，需要盡量為乾淨的純色，而為了防止物品在運輸過程中傾倒掉落車外，物架的四周邊緣須為傾斜形狀，使物體在掉出邊緣前能夠滑回物架中心，因此我們使用剩餘的白色塑膠瓦楞板，剪裁拼接為托盤狀（如圖 4-6 所示），使其滿足載貨的需求。

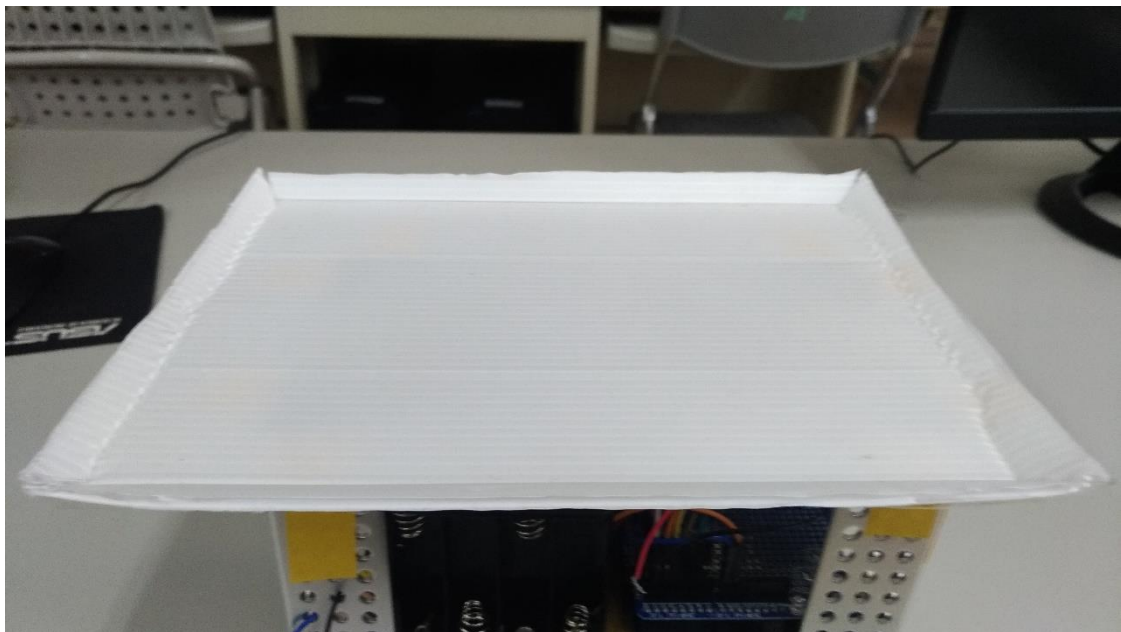


圖 4-6 自走車物架圖

4.4.3 感測器設置：

自走車在行進過程間，若是遇到停靠點的柵欄或是產線上所突然闖入的障礙物，自走車需要即時停止來避免相撞，因此我們決定在車架的車頭處加裝 HC-SR04 超音波感測器來完成此任務。

而循線感測器的部分，起初我們使用三個紅外線感測器來進行循線（分別放置於車體的左邊、中間、右邊），只要三個感測器之一偵測到黑線，自走車才會做動（如圖 4-7 所示），然而實際運作時，三個感測器常因為相隔太遠而讓軌跡跑至三個感測器之間的感測盲區，導致自走車偵測不到軌跡而停止；但相隔太近時又無法感測到轉彎幅度大的軌跡，因此我們決定使用第二種循線方案，捨去了車體中間的感測器，將感測器縮減為兩個，使軌跡始終保持在車體中間，一旦左右兩側的感測器偵測到軌跡黑線邊緣，自走車就會往反方向進行補正（如圖 4-8 所示）。

使用新的感測方案後自走車便能於產線中正常運作，且感測器的數量越少，程式中感測器的判斷式就能越短、自走車內空間以及主板的可使用腳位就越多、主板工作要求電壓就能越少，可謂一舉數得。

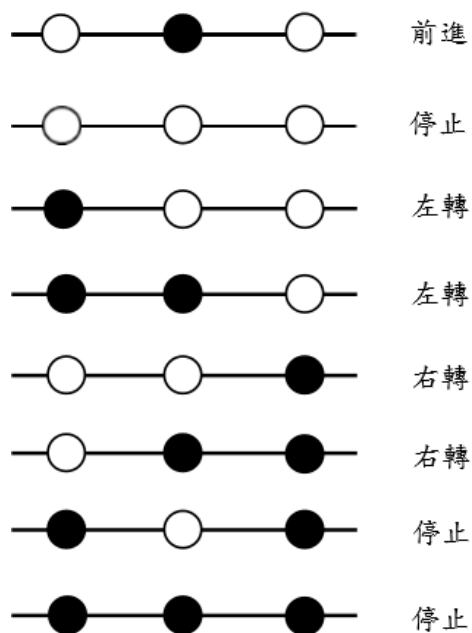


圖 4-7 最初的循線方案

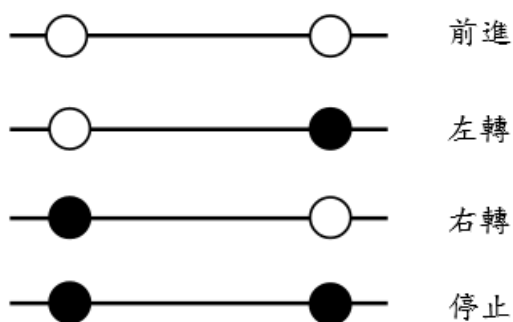


圖 4-8 第二種循線方案

4.4.4 車體佈線：

在此部份中需要格外謹慎，若在此接線過程中稍有疏失，那將會帶來不必要的經濟損失甚至安全上的危險，其接線方式如下：

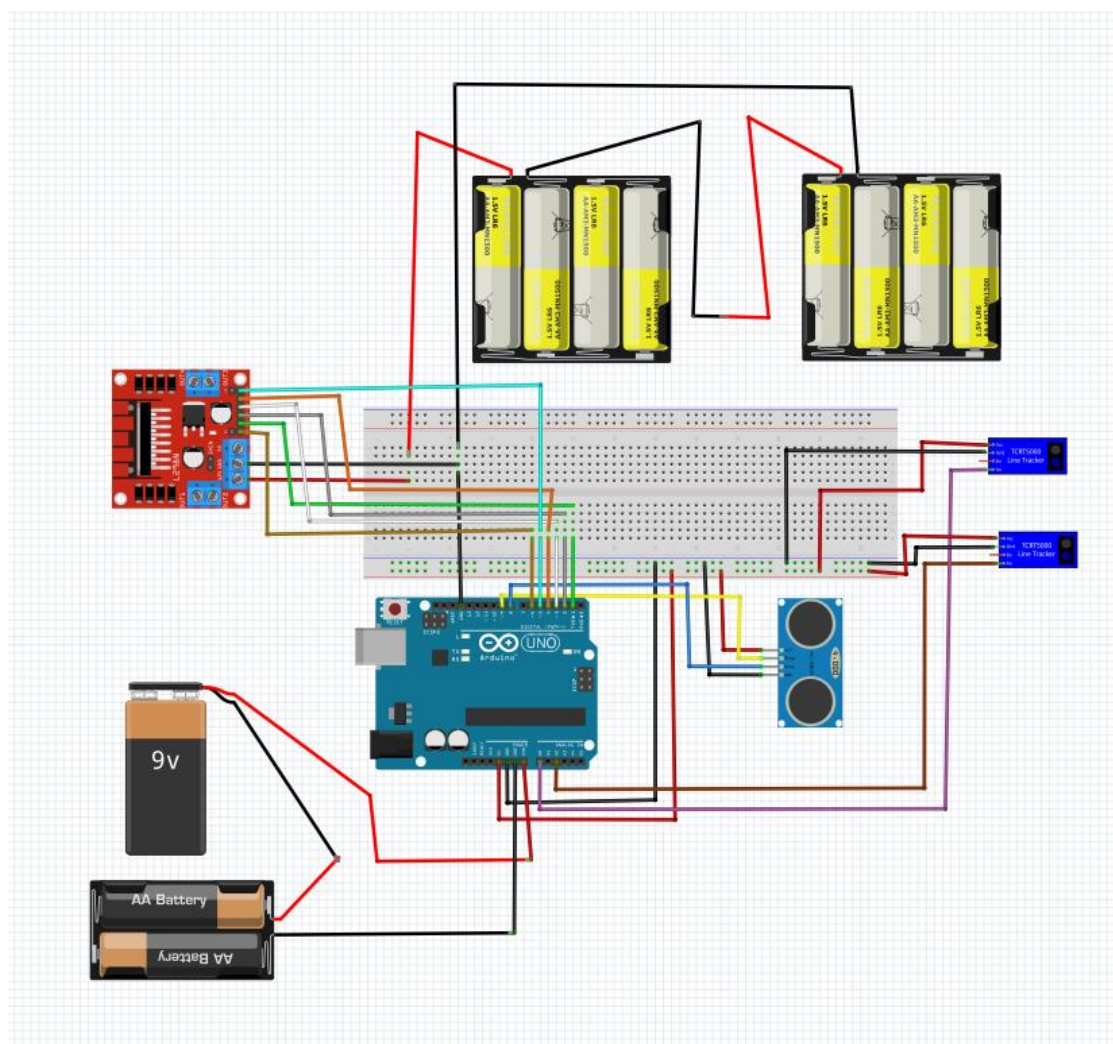


圖 4-9 自走車接線示意圖（使用 fritzing 軟體繪製）

第五章 實驗結果與展示

5.1 產線完成圖：

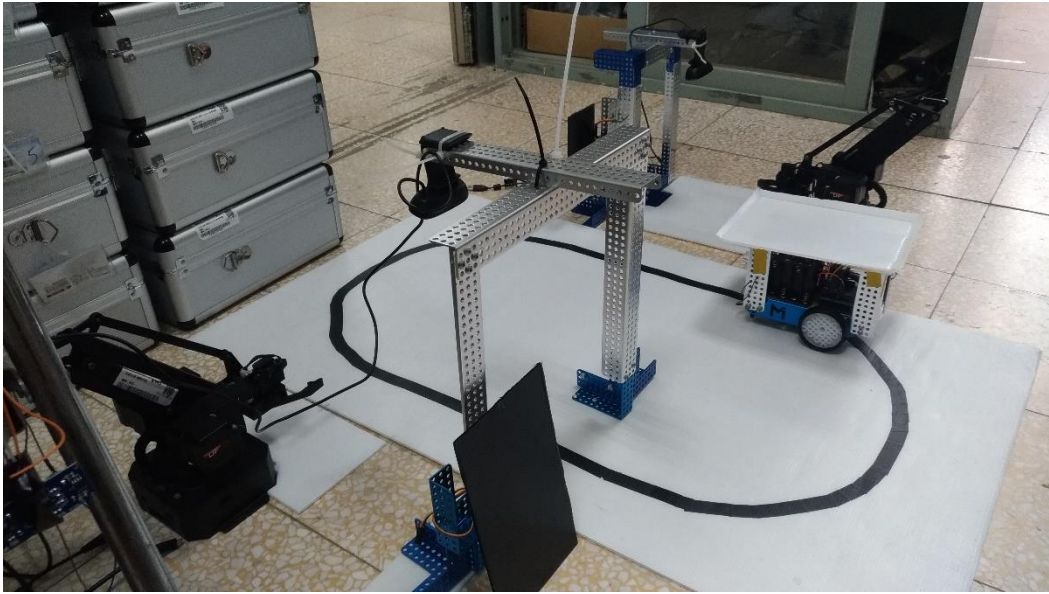


圖 5-1 產線系統完成圖

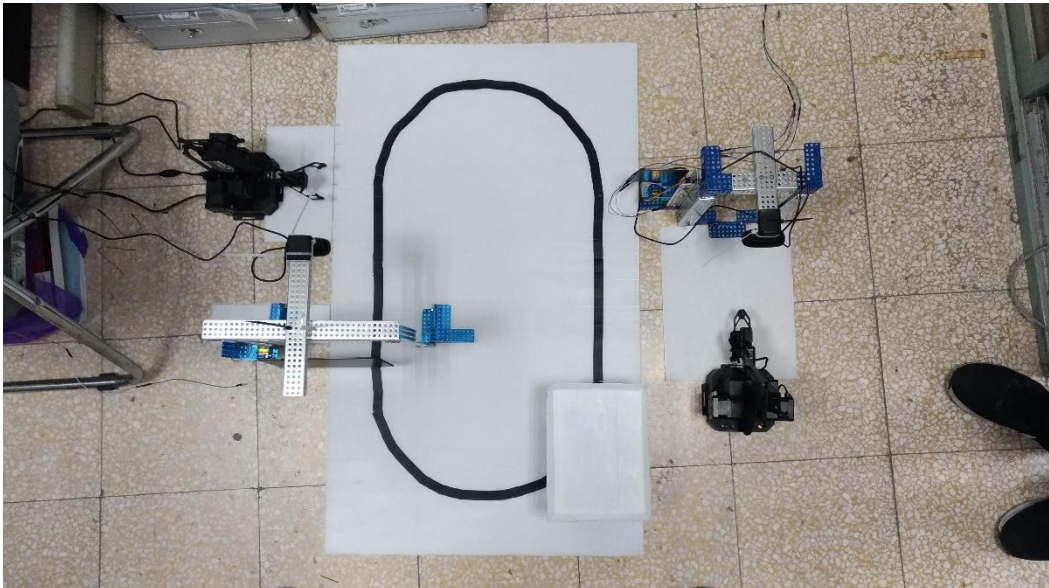


圖 5-2 產線系統完成上視圖

第六章 未來展望

本次專題中於自動化產線之發展已有些微成果，然而尚有相當大的發展空間，將來若是實戰運用在大型工廠生產線之中，那麼想必須要擴充更多的內容，例如在各機電配合之間加入可程式化邏輯控制器（PLC），又或者於機械視覺系統之中加入機械學習之概念，使其能夠應對更多樣的狀況、在產線之中擁有更強的適應力。而產線中的無人運載車的體積也會隨著更多、更靈敏、更強力的感測器以及馬達而變得更大，其載重能力也會隨之增強。在產線系統中也缺少著人性化的人機介面，使其能夠讓產線管理著隨時以輕易的方式來設定甚至於客製化整個產線系統。在不遠的將來也許自動化生產線就會被運用至人類無法抵達的惡劣環境之中進行作業，到那時整個人類工業社會的自動化產線系統想必會發展至我們現今無法想像之地步，這些都給予了我們宏大的遐想與發展空間，值得我們將來去探索。

第七章 結論

在取得了實際運作成果之後，我們深深得體悟到了自動化工業於現今的重要性，尤其當前全世界皆正忍受著新型冠狀肺炎病毒(covid-19)的肆虐及攻擊之情況下，在疫情尚無法有效獲得控制前，人類為維護自身生命的安全，全世界的國家都被迫採取嚴格的隔離政策以避免人員群聚相互感染進而加重疫情災害，如此措施導致工廠必須停工，導致各產業鏈出現生產斷面，造成全世界經濟嚴重衰退，許多工廠面臨倒閉風險，各國政府為振興經濟所挹注資金再加上疫情所造成的損失，金額簡直無法估計，試想全世界未來均有可能再遭遇類此疫病或類似之狀況，那麼各國政府機構為控制並減少如此無可避免的巨大損失，想必會將更多的精力投資在工廠無人化發展之上，此情勢發展正符合本次專題製作之意向。

參考文獻：

- [1]<https://www.playrobot.com/arm/1781-uarm-swift-pro-blockly-ufactory.html>
- [2]<https://makerpro.cc/2018/11/opencv-background-subtractor/>
- [3]<https://smasoft-tech.com/tag/%E5%BD%B1%E5%83%8F%E5%89%8D%E8%99%95%E7%90%86%E3%80%81%E5%BD%B1%E5%83%8F%E9%A0%90%E8%99%95%E7%90%86%E3%80%81%E6%A9%9F%E5%99%A8%E8%A6%96%E8%A6%BA%E3%80%81%E4%BA%8C%E5%80%BC%E5%8C%96%E3%80%81%E5%BD%A2-2/>
- [4]<https://www.itread01.com/content/1543921874.html>
- [5]<https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/560259/>
- [6]<https://medium.com/@pomelyu5199/canny-edge-detector-%E5%AF%A6%E4%BD%9C-opencv-f7d1a0a57d19>
- [7]<https://chtseng.wordpress.com/2016/12/05/opencv-contour%E8%BC%AA%E5%BB%93/>
- [8]<https://blog.everlearn.tw/%E7%95%B6-python-%E9%81%87%E4%B8%8A-raspberry-pi/raspberry-pi-3-model-b-%E8%88%87-hc-sr04-%E8%B6%85%E9%9F%B3%E6%B3%A2%E6%84%9F%E6%B8%AC%E5%99%A8%E4%B9%8B%E6%87%89%E7%94%A8>

附錄 A

手臂程式：

```
import RPi.GPIO as GPIO

import time, random

from uarm.wrapper import SwiftAPI

from cv2 import cv2

swift = SwiftAPI(filters={'hwid': 'USB VID:PID=2341:0042'}, cmd_pend_size = 2)

swift.waiting_ready()

device_info = swift.get_device_info()

firmware_version = device_info['firmware_version']

if firmware_version and not firmware_version.startswith(('0.', '1.', '2.', '3.')):

    swift.set_speed_factor(0.00005)

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

trig , echo = 25, 8

motor_pin, PWM_FREQ, STEP = 17, 50, 90
```



```
GPIO.setup(trig, GPIO.OUT)
```

```
GPIO.setup(echo, GPIO.IN)
```

```
GPIO.setup(motor_pin, GPIO.OUT)
```

```
pwm = GPIO.PWM(motor_pin, PWM_FREQ)
```

```
pwm.start(90)
```

```
def angle_to_duty_cycle(angle = 0):
```

```
    duty_cycle = (0.05 * PWM_FREQ) + (0.19 * PWM_FREQ * angle / 180)
```

```
    return duty_cycle
```

```
def get_distance():
```

```
    GPIO.output(trig, False)
```

```
    time.sleep(0.0005)
```

```
    GPIO.output(trig, True)
```

```
    time.sleep(0.001)
```

```
    GPIO.output(trig, False)
```

```
    while GPIO.input(echo) == 0:
```

```
        start = time.time()
```

```

while GPIO.input(echo) == 1:

    end = time.time()

    return (end - start) * 17150

swift.set_position(x = 110, y = 0, z = 20, speed = 9e7)

time.sleep(3)

while True:

    if 4 < get_distance() <= 20:

        dc = angle_to_duty_cycle(0) #

        pwm.ChangeDutyCycle(dc)

        time.sleep(5)

        cap  = cv2.VideoCapture(0)

        while True:

            ret, frame = cap.read()

            cv2.imshow("capture", frame)

            if cv2.waitKey(5000):

                cv2.imwrite("image.jpg", frame)

                break

```

```

cap.release()

cv2.destroyAllWindows("capture")

image = cv2.imread("image.jpg")

white_board = cv2.imread("white_board.jpg")

diff = cv2.absdiff(white_board, image)

gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

ret, th1 = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

blur = cv2.GaussianBlur(th1, (11, 11), 0)

Canny = cv2.Canny(blur, 50, 150)

_, find_contours, hierarchy = cv2.findContours(Canny,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

image_contours = cv2.drawContours(image, find_contours, -1, (0, 0, 255),
1)

cv2.imshow("image_contours", image_contours)

cv2.waitKey(5000)

cv2.destroyAllWindows("image_contours")

```

p = 1

for contours1, contours2 in zip (range(len(find_contours)), find_contours):

 epsilon = 0.025 * cv2.arcLength(find_contours[contours1], True)

 approx = cv2.approxPolyDP(find_contours[contours1], epsilon, True)

 corners = len(approx)

 if corners == 3:

 shape_type = "triangle"

 elif 4 <= corners <= 6:

 shape_type = "rectangle"

 elif 7 <= corners <= 15:

 shape_type = "round"

 else:

 shape_type = "無法辨識"

 M = cv2.moments(contours2)

 if M["m00"] == 0:

 M["m00"] = 1

```

cx = int(M["m10"] / M["m00"])

cy = int(M["m01"] / M["m00"])

x, y, w, h = cv2.boundingRect(contours2)

cv2.circle(image_contours, (cx, cy), 3, (0, 255, 0), -1)

cv2.putText(image, str(p), (x - 10, y + 10),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

cv2.putText(image, str(shape_type), (x + 20, y + 10),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2) # 給識別物件寫上 shape_type

p += 1

print("形狀: %s, 鏡頭內中心座標: (%d px, %d px), 絕對座標:
(%d, %d)%"

(shape_type, cx, cy, round(cy * 0.25363 + 135, 3), round(cx *
0.25455 - 67, 3)))

cv2.imshow("image_circle_contours", image_contours)

cv2.waitKey(5000)

cv2.destroyAllWindows()

```

```
def repeat_action():

    random_value = random.random() * 10

    swift.set_position(y = 200 + random_value, speed = 9e7)

    time.sleep(3)

    swift.set_position(z = 150, speed = 9e7)

    time.sleep(3)

    print(swift.set_gripper(catch = False))

    time.sleep(3)

    swift.set_position(z = 160, speed = 9e7)

    time.sleep(3)

    swift.set_position(y = 0, speed = 9e7)

    time.sleep(3)

    swift.set_position(z = 70, speed = 9e7)

    time.sleep(3)

    swift.set_position(z = 50, speed = 9e7)

    time.sleep(3)

    swift.set_position(z = 20, speed = 9e7)
```

```

time.sleep(3)

swift.set_position(x = 110, speed = 9e7)

time.sleep(3)

A = [cv2.moments(contours) for contours in find_contours]

cx = [int(M["m10"] / M["m00"]) for M in A]

cy = [int(M["m01"] / M["m00"]) for M in A]

for contours1 in range(len(find_contours)):

    speed = 9e7    # x=110~260, y=300(左)~-300(右), z=20~170

    swift.set_position(y = cx * 0.25455 - 67, speed = 9e7)

    time.sleep(3)

    swift.set_position(z = 10, speed = 9e7)

    time.sleep(3)

    swift.set_position(x = cy * 0.25363 + 135, speed = 9e7)

    time.sleep(3)

    print(swift.set_gripper(catch = True))

    time.sleep(3)

```

```

swift.set_position(z = 160, speed = 9e7)

time.sleep(3)

epsilon = 0.09 * cv2.arcLength(find_contours[contours1], True)

approx = cv2.approxPolyDP(find_contours[contours1], epsilon, True)

corners = len(approx)

if corners == 3:

    swift.set_position(x = 185, speed = 9e7)

    time.sleep(3)

    repeat_action()

elif corners == 4:

    swift.set_position(x = 175, speed = 9e7)

    time.sleep(3)

    repeat_action()

elif 5 <= corners <= 15:

    swift.set_position(x = 165, speed = 9e7)

    time.sleep(3)

```



```
repeat_action()

else: # shape_type = "無法辨識"

    swift.set_position(x = 155, speed = 9e7) # 移動到放置點 X

    time.sleep(3)

    repeat_action()

    dc = angle_to_duty_cycle(90)

    pwm.ChangeDutyCycle(dc)

    time.sleep(5)

else:

    print("沒有車子經過")
```

附錄 B

樹莓派整體安裝指令：

```
wget http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-armv7l.sh
```

```
sudo md5sum Miniconda3-latest-Linux-armv7l.sh
```

```
sudo /bin/bash Miniconda3-latest-Linux-armv7l.sh
```

```
/home/pi/miniconda3
```

```
sudo nano /home/pi/.bashrc
```

```
export PATH="/home/pi/miniconda3/bin:$PATH"
```

```
sudo apt-get install spyder3
```

```
sudo chown -R pi /home/pi/miniconda3
```

```
conda config --add channels rpi
```

```
conda install python=3.6
```

```
conda install -y -c conda-forge opencv
```

```
pip3 install opencv-python==3.3.0.10
```

附錄 C

```
const int In1 = 1;
```

```
const int In2 = 2;
```

```
const int In3 = 3;
```

```
const int In4 = 4;
```

```
const int sensor_left = A2;
```

```
const int sensor_right = A0;
```

```
const int trig = 9;
```

```
const int echo = 8;
```

```
const int trigdistance = 12;
```

```
long duration, cm;
```

```
int SLL;
```

```
int SRR;
```

```
int SensorStatus;
```

```
int ENA = 6;
```

```
int ENB = 5;
```

```
void setup() {  
  
    Serial.begin(9600);  
  
    pinMode(In1, OUTPUT);  
  
    pinMode(In2, OUTPUT);  
  
    pinMode(In3, OUTPUT);  
  
    pinMode(In4, OUTPUT);  
  
    pinMode(ENA, OUTPUT);  
  
    pinMode(ENB, OUTPUT);  
  
    pinMode(sensor_left, INPUT);  
  
    pinMode(sensor_right, INPUT);  
  
    pinMode(trig, OUTPUT);  
  
    pinMode(echo, INPUT);  
  
    pinMode(A4, INPUT);  
  
    pinMode(A3, INPUT);  
  
}
```

```

void loop() {

    int pot_r = analogRead(A3);

    int pot_l = analogRead(A4);

    int potvalue_r = map(pot_r, 0, 1023, 0 , 255);

    int potvalue_l = map(pot_l, 0, 1023, 0 , 255);

    Serial.print("r = ");Serial.print(potvalue_r);

    Serial.print("l = ");Serial.println(potvalue_l);

    int SLL = 0 ;

    int SRR = 0 ;

    int SensorStatus = 0;

    digitalWrite(trig, LOW);

    delayMicroseconds(50);

    digitalWrite(trig, HIGH);

    delayMicroseconds(100);

    digitalWrite(trig, LOW);

    pinMode(echo, INPUT);

```

```

duration = pulseIn(echo, HIGH);

cm = (duration/2) / 29.1;

if(cm <= trigdistance){

    car_stop();

}

else{

SLL = analogRead(A2);

if(SLL>200){

    SensorStatus = (SensorStatus + 1);

}

SRR = analogRead(A0);

if(SRR>200){

    SensorStatus = (SensorStatus + 2);

}

```

```
switch(SensorStatus){  
  
    case 0:  
  
        car_front();  
  
        break;  
  
    case 1:  
  
        car_left();  
  
        break;  
  
    case 2:  
  
        car_right();  
  
        break;  
  
    }  
  
}  
  
}
```

```
void car_front(){

    int pot_r = analogRead(A3);

    int pot_l = analogRead(A4);

    int potvalue_r = map(pot_r, 0, 1023, 0 , 255);

    int potvalue_l = map(pot_l, 0, 1023, 0 , 255);

    digitalWrite(In1, HIGH);

    digitalWrite(In2, LOW);

    digitalWrite(In3, LOW);

    digitalWrite(In4, HIGH);

    analogWrite(ENA, 255);

    analogWrite(ENB, 90);

}
```

```
void car_stop(){

    digitalWrite(In1, LOW);

    digitalWrite(In2, LOW);

    digitalWrite(In3, LOW);

    digitalWrite(In4, LOW);

}
```



```

    analogWrite(ENA, 0);

    analogWrite(ENB, 0);

}

void car_right(){

    int pot_r = analogRead(A3);

    int pot_l = analogRead(A4);

    int potvalue_r = map(pot_r, 0, 1023, 0 , 255);

    int potvalue_l = map(pot_l, 0, 1023, 0 , 255);

    digitalWrite(In1, HIGH);

    digitalWrite(In2, LOW);

    digitalWrite(In3, HIGH);

    digitalWrite(In4, LOW);

    analogWrite(ENA, 150);

    analogWrite(ENB, 90);

}

void car_left(){

    int pot_r = analogRead(A3);

```

```
int pot_l = analogRead(A4);

int potvalue_r = map(pot_r, 0, 1023, 0 , 255);

int potvalue_l = map(pot_l, 0, 1023, 0 , 255);

digitalWrite(In1, LOW);

digitalWrite(In2, HIGH);

digitalWrite(In3, LOW);

digitalWrite(In4, HIGH);

analogWrite(ENA, 90);

analogWrite(ENB, 150);

}
```

附錄 D

裝貨區手臂座標計算：

手臂於鏡頭左上原點(135 , -67)，特定點座標(170 , 3)

鏡頭特定點座標：(275 , 138)

$$\text{X 軸比例} : \frac{170-135}{138} = 0.25363$$

$$\text{Y 軸比例} : \frac{3-(-67)}{275} = 0.25455$$

$$\text{驗算} : 275 \times 0.25455 + -67 = 3.00125 \cong 3$$

卸貨區手臂座標計算：

因擺放位置關係，鏡頭 X 軸原點改於右上角(639-327=312)

$$\text{X 軸比例} : 250 - 312 \times A = 185, A = 0.2083$$

$$\text{Y 軸比例} : \frac{-120-(-60)}{299} = -0.20066$$

附錄 E

