基於深度學習技術搜索生還者之聲控探測車

學生：邱妤靜、李欣恩、陳遠浩

指導教授：朱守禮

目錄

圖目錄

第一章　緒論

1.1 摘要

科技隨時代不斷演進，如今勞動作業走向機械化，救災方面亦如此，因此我們將研究方向定於災區環境與生還者搜索，以期協助救災人員執行現場工作。

本專題核心為實作探測車，包含地形探勘、搜索生還者及地圖繪製三項功能。藉探測車體積小優勢，搜索救災人員較難進入空間，並將障礙物座標化，以期增加搜尋效率。透過其上裝設之攝影機、找尋空隙之演算法，以物件辨識方式搭配超音波測距，實現影像避障功能，也將結合物件辨識與超音波避障，以利搜索生還者。另將使用深度學習技術建立及訓練物件辨識模型，並提供語音命令功能。

1.2 研究動機

日趨科技化的現今社會，人們對機器的仰賴愈來愈高，除提高工作效率、增加內容正確性外，還可代替人類執行高危險、高準確、高重複之工作。而多數災害現場危險又混亂，許多救災人員須暴露於危險環境中，若遇通訊狀況不佳、地形崎嶇等情況，更使救災任務困難重重。因此本專題針對救災部分深入研究。

我們欲設計一探測車，代替救災人員對未知災區進行初步探測、回傳即時影像、於探測過程中尋找生還者並繪製地圖，以期增加搜索效率。由於探測車需於崎嶇道路上搜索，因此本專題車體將模擬現實中較易克服此問題之履帶車體，且因應探測車需在現場自行走動，偵測障礙物並閃避成為本專題重點之一。偵測同時，我們更期望擁有搜索受難者功能，因此將使用人臉辨識進行搜索，且為使救災人員能順利執行救援任務，將即時回傳受難者、沿途障礙物之座標。另考量救災人員進入現場時，會攜帶大量救援器具或搬運受難者等雙手無法活動裝況，除基本遙控器輸入命令，我們欲增加語音指令，如此救災人員可以口頭方式給予探測車指令。綜合上述以期降低救災人員受傷機率、降低救災時間、增加受難者於黃金救援時間內獲救機會，將人力耗損及風險降到最低。

第二章　車體安裝與軟硬體

2.1 車體安裝與問題

本專題所用之探測車為以直流馬達帶動車體之履帶車，其上加裝 Raspberry pi 3 做為控制車體元件，另加裝單顆超音波搭配伺服馬達於車頭正中央，實現 180 度偵測功能輔助避障，未來欲增設超音波於伺服馬達兩側，增加避障精確度。

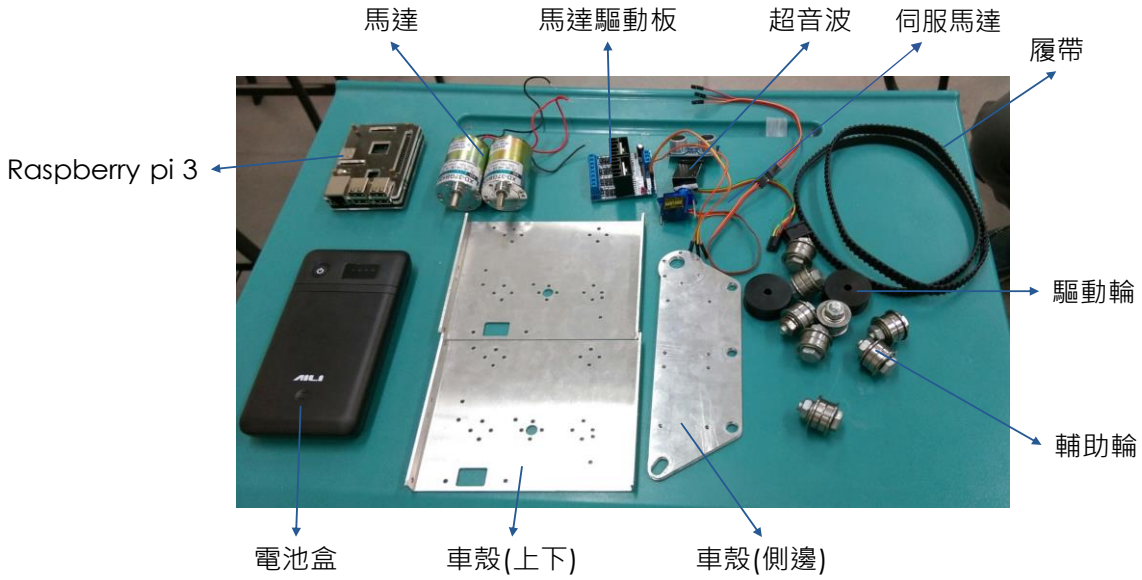
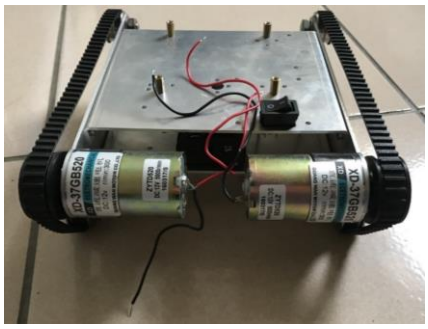


圖 1、自走車零件



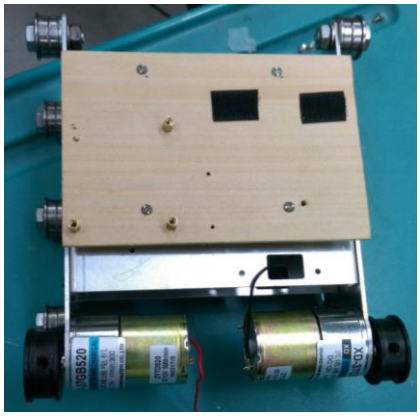


圖 2、車體組裝
(上為車體外觀、下為加裝木板)

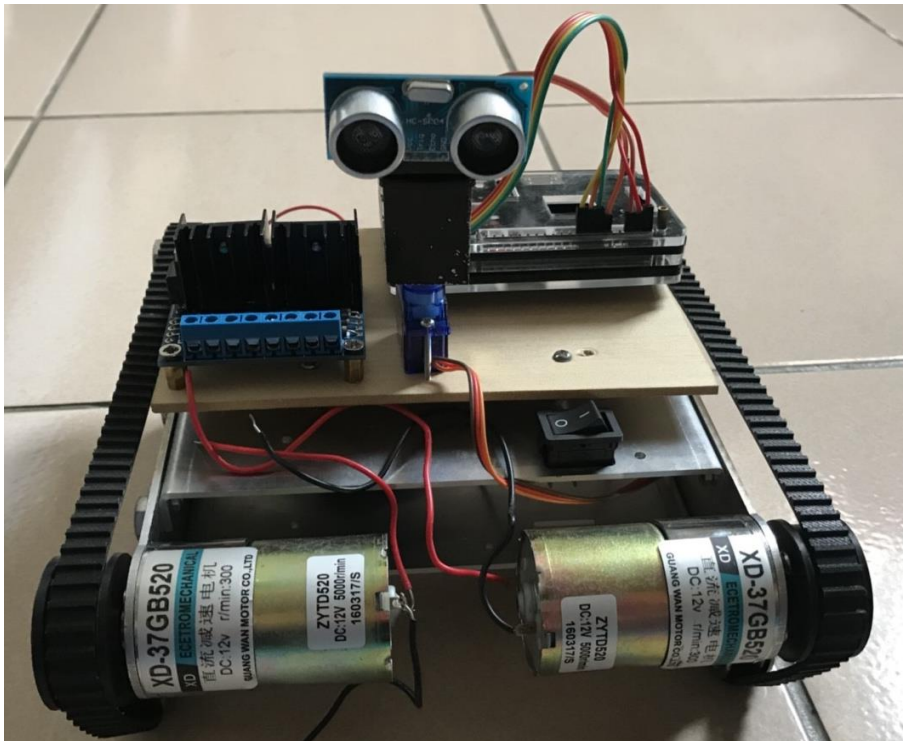


圖 3、加裝完畢各元件之履帶車體

組裝期間因馬達和側邊車板連接之鑽孔與所附螺絲大小不合，導致兩者不易貼合，後來改尋較適合之螺絲替代；組裝後因驅動輪與輔助輪不平行，導致行進時履帶易跑軌，我們於各輔助輪上加裝墊片，使其維持平行狀態；後來發現行進於磨擦力較高的地面時馬達無法帶動履帶，我們多次重鎖輔助輪以調整履帶鬆緊度，但效果不彰，最後於驅動輪加綁橡皮筋才解決與履帶間摩擦力不足問題。

2.2 使用之軟硬體介紹

本專題主要用到 Raspberry Pi 3 與 TensorFlow，以 Raspberry Pi 3 為主要控制主機，在其上裝設 TensorFlow，TensorFlow 用來訓練辨識所使用的模型，Raspberry Pi 3 則利用訓練好的模型做推論。

A. Raspberry Pi 3[1]

Raspberry Pi 3 是一款基於 Linux 的單板電腦，擁有一個 Ethernet、4 個 USB 介面、以及 HDMI（支援聲音輸出）和 RCA 端子輸出支援。以 Python 作為主要程式設計語言，Raspberry Pi 3 首次將無線網路與藍牙功能內建於主機之中，記憶體容量為 1G，相較於 1 代與 2 代 Raspberry Pi 採用的單核心 ARM1176JZF-S（700MHz）與四核心 ARM Cortex-A7（900MHz），Raspberry Pi 3 採用 64 位元四核心 ARM Cortex-A53（1.2GHz），其效能比起第一代 Raspberry Pi 可提升 10 倍。本專題將使用 Raspberry Pi 3 作為主要的控制電腦，做物件辨識之推論並控制自走車行為。

B. TensorFlow[2]

TensorFlow 是一個由 Google 提供的開放原始碼程式庫，適用於各種感知和語言理解的機器學習。支持不少針對行動端訓練和推論之深度學習模型，包括視覺模型、圖片識別模型和裝置對話模型等。

其具備跨平台的功能，可以在 Windows、iOS、Linux 等平台上執行，本專題即在 Raspberry Pi 3 上執行 TensorFlow。TensorFlow 也支持多種語言撰寫，但對 Python 的支援最好，本專題也是用 Python 撰寫程式。

TensorFlow 由 Tensor 與 Flow 組成，Tensor 指的是「張量」，為一種幾何實體，0 維的張量維純量，1 維的張量為向量，2 為以上的張量為矩陣。Flow 則是指「資料處理的流程」，TensorFlow 計算時都是以圖來計算，此種圖稱為「計算圖(computation graph)」。本專題中將使用 TensorFlow 建立並訓練物件辨識的模型。

第三章　超音波避障

本專題之探測車其上裝設伺服馬達與單顆超音波，擁有 180 度偵測功能，將使用影像避障搭配超音波避障。本章將說明專題中所使用之超音波避障方法。

3.1 超音波測距原理

超音波為自走車在進行避障時，常使用的測距系統之一，其優點為不受光線、粉塵、煙霧干擾，並可搭配伺服馬達實現多方向偵測，以下為其原理。

根據[3]所述，超音波感測器由超音波發射器、接收器和控制電路組成。當它被觸發時，會發射 40 kHz 的聲波並從離它最近的物體接收回音(如圖 4所示)。

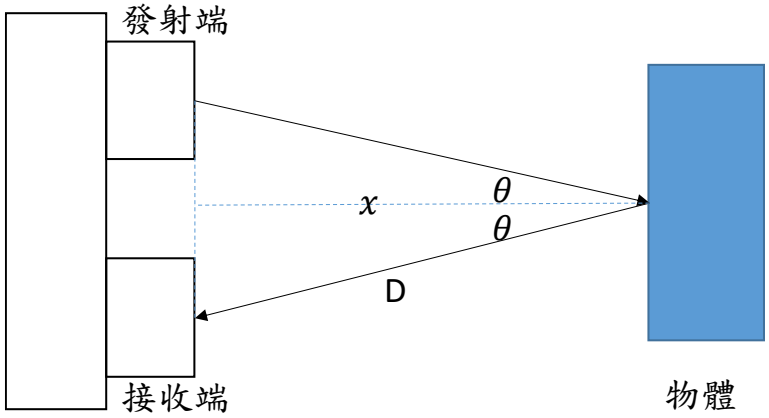


圖 4、超音波感測器聲波示意圖

聲音在空氣中的傳播速度會受到溫度影響，見(1)：

$v = 331 + 0.6T\ (m/s)$ ----------(1)

其中 v 為聲音在空氣中的傳播速度，T 為環境溫度(℃)。

圖 1 中 D 為聲波行經路徑長度，見(2)：

$D = v \times \frac{\Delta t}{2}$，其中$\Delta t$為超音波發射到接收所需時間 -------(2)

圖 1 中θ為超音波入射角，但因發射端與接收端距離相對於x極小，因此角度θ可忽略不計，故與物體之距離x近似於 D。

$x = D\cos\theta = v \times \frac{\Delta t}{2}\cos\theta \approx v \times \frac{\Delta t}{2}$ ----------(3)

因聲音在空氣中傳速約等於 340 m/s，本文中取 v 為 340 m/s。

因災區較易遇光線不足、粉末等干擾，因此選擇超音波期望克服此類問題。

3.2 PWM 與伺服馬達之關係

由於使用單顆超音波避障，為增加避障的準確度，需量測多角度距離，我們調整 PWM 的 Duty Cycle，控制伺服馬達之轉動角度，本文中所使用之伺服馬達為 SG90(如圖 5 所示)，其旋轉角度為0°~180°(逆時針旋轉)，大多數的伺服馬達以 1 ms 的脈衝寬度當作180°，而 2 ms 則為0°，也就是用脈衝寬度 1 ms~2 ms 可以表示出0°~180°的旋轉角度。

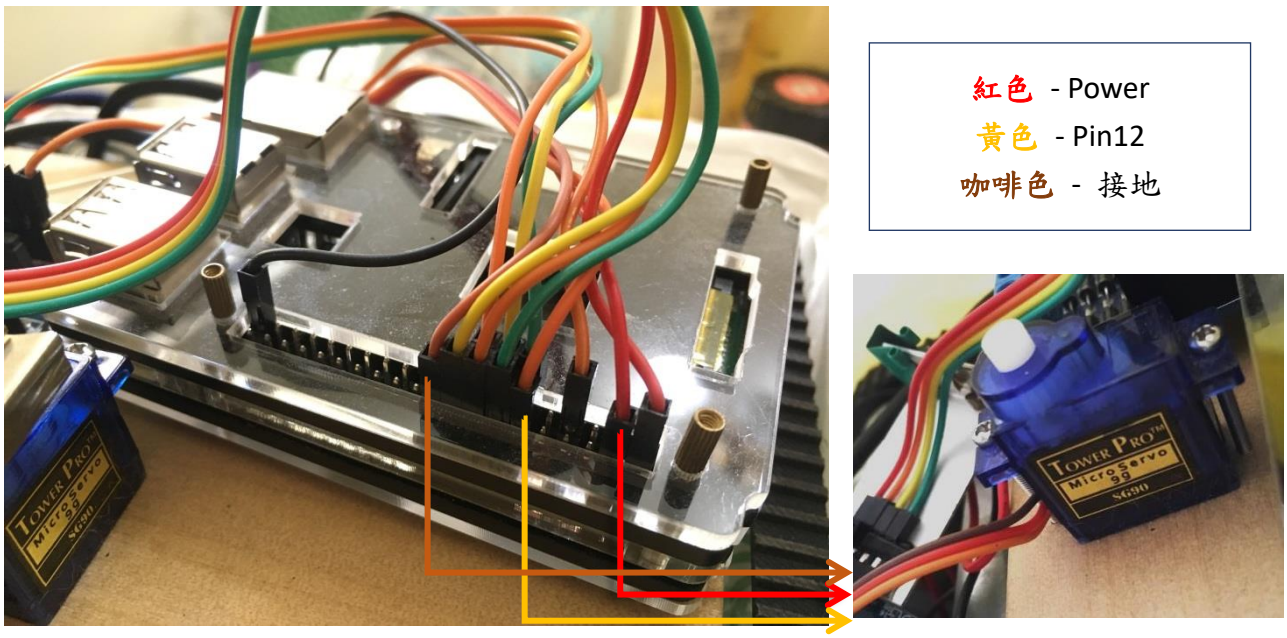


圖 5 、伺服馬達與 Raspberry Pi 3 之接線

然而 SG90 預設0°的脈衝寬度是 0.5 ms，而預設180°的脈衝寬度則是 2.4 ms，表示 SG90 的旋轉角度介於 0.5 ms~2.4 ms。

SG90 的工作頻率為 50 Hz，當頻率為 50 Hz 時，週期為 20 ms，根據[3]則：

0.5 ms 脈衝寬度的 Duty Cycle $\frac{0.5}{20} \times 100\% = 2.5\%$ ----------(4)

2.4 ms 脈衝寬度的 Duty Cycle $\frac{2.4}{20} \times 100\% = 12\%$ ----------(5)

由(4)、(5)可知把 2.5%~12% 之間的差距平分給 180 個角度，就可以控制伺服馬達的旋轉角度，公式如下：

當角度為x時的工作週期$(\%) = 2.5 + \frac{12-2.5}{180} \times x$ ---------- (6)

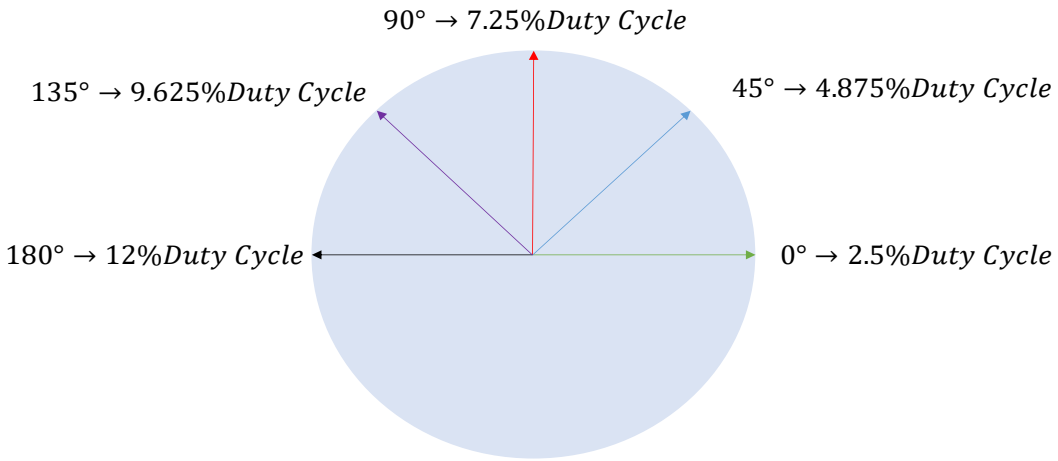


圖 6、角度與 PWM Duty Cycle 示意圖

3.4 實驗階段

A. 階段一：僅利用測距避障，不考慮旋轉角度

此階段只以測距為考慮因素，決定車子要直走、後退、轉彎，控制機制如下：

首先先測前方距離，並以所測之距離座三種判斷。

(1) 若前測距大於 30 公分，則往前走所測距離的一半。

(2) 若前測距小於等於 10 公分，則後退。

(3) 若前測距小於等於 30 公分，則測左右距離決定要左轉還是右轉。若左邊距離大於右邊距離，則左轉；反之，則右轉。

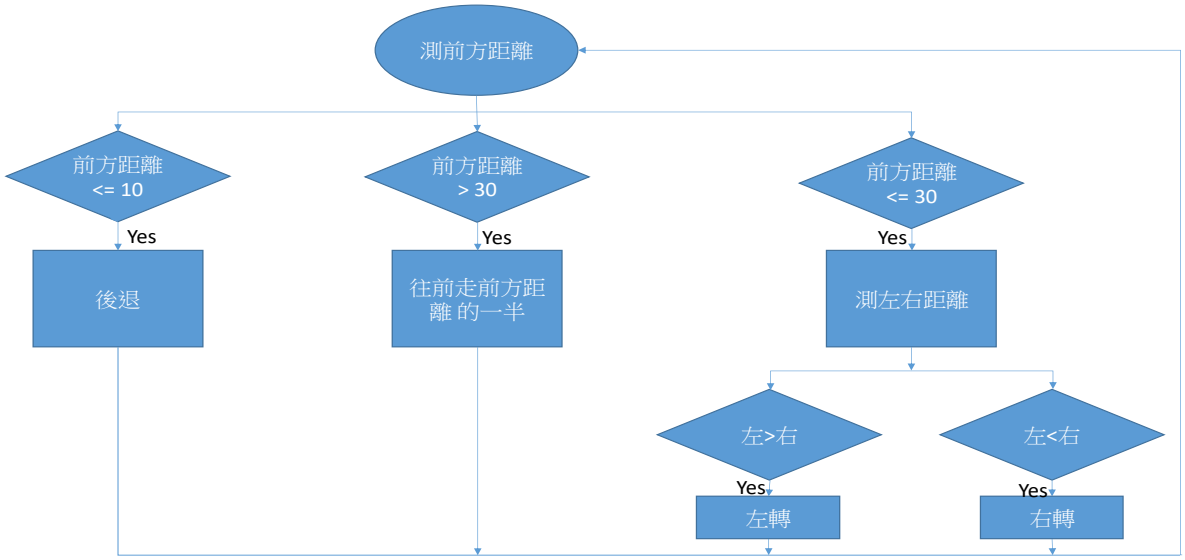


圖 7、階段一流程圖

此階段由於超音波未測左右前方，只能避開位於車子正前方之障礙物，且因此若障礙物的位置如圖 8 時，車子前進時仍會撞上。

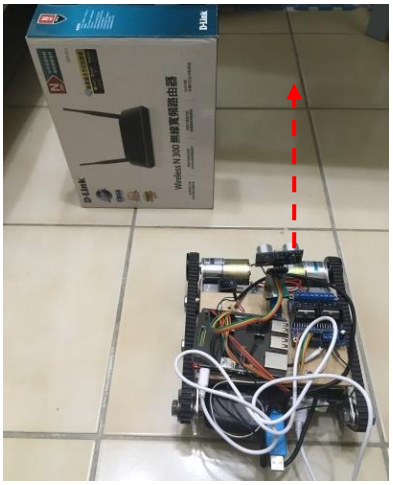


圖 9、障礙物與車子實際位置圖

對於超音波而言，前方是沒有障礙物的，此時車子前進，左側會撞上障礙物。

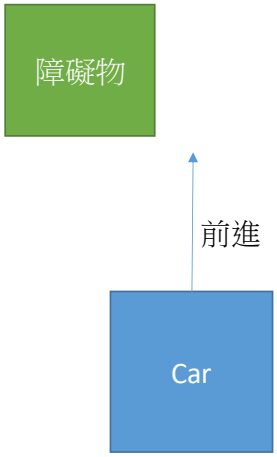


圖 8、障礙物與車子位置示意圖(一)

B. 階段二：超音波多角度測距，並指定轉彎角度

此階段主要解決當依車子前測距判斷可前進時，是否仍有左前或右前肢障礙物存在，此階段之超音波增加多角度的測距，所測角度由右到左分別為0°、30°、60°、90°、120°、150°、180°共七個方向，以90°為中心點(前方)，分別判斷右前方與左前方是否可行，並由左、右前方之測距微調車子旋轉角度。

以下為此階段之細部調整階段：

B.1 提前預知障礙物存在與否：

首先測量半車身長度為 10 公分，且測距 20 代表有障礙物，又因超音波於 40 公分內測距較準確，我們取超音波所測距離(除前測距以外)為斜邊，依畢氏定理及三角函數，可得若車子要前進，前方與左、右至少夾角為 14.5 度且斜邊為 40 公分內需無障礙物，如圖 10 所示。之後增加夾角度數，算出理想斜邊值，與實際測量值比較，若理想值>測量值，表示有障礙物。

理想斜邊值計算：$\frac{10}{\sin\theta}$

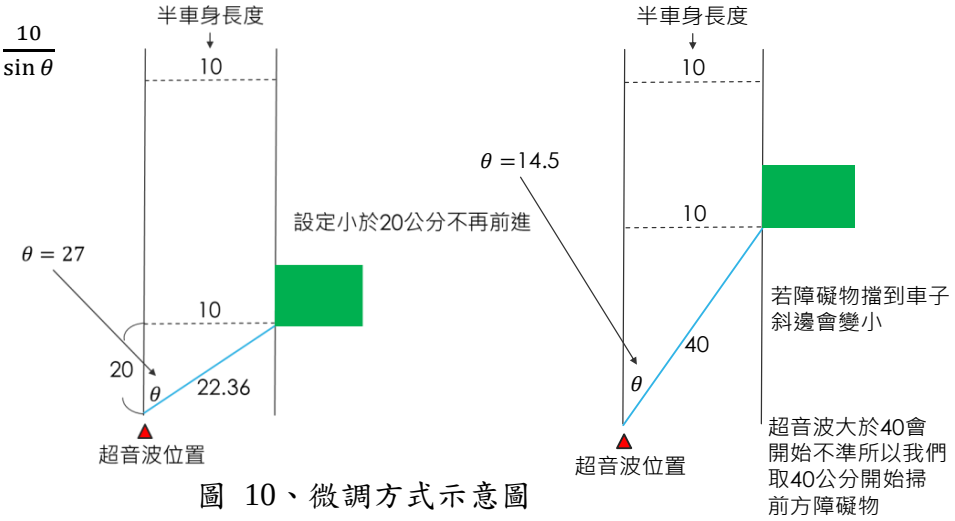


圖 10、微調方式示意圖

判斷左前、右前是否有障礙物：

FindLeftSpace()：

Angle = 90 + 14.5 //測左邊角度

while(Angle <= 117)：

MotorTurnAngle = $0.05 \times 50 + \frac{0.19\times50\times Angle}{180}$

Edge = $\frac{10}{\sin(Angle)}$ //理想值

leftDist = 超音波測距 //測量值

if (leftDist < Edge)：return True

Angle = Angle + 4

return Fals

從左邊開始測，角度從 14.5 開始(104.5 = 90+14.5)，然後進迴圈，先算出斜邊距離(理想值)，在與超音波測到的距離作比較(測量值)，若測量值 < 理想值，則表示左邊有障礙物擋到。

FindRightSpace()：

Angle = 90 - 14.5 //測右邊角度

while(Angle >= 63)：

MotorTurnAngle =

$0.05 \times 50 + \frac{0.19\times50\times Angle}{180}$

Edge = $\frac{10}{\sin(Angle)}$ //理想值

rightDist = 超音波測距 //測量值

if (rightDist < Edge)：return True

Angle = Angle - 4

return False

右邊同左

圖 11、階段二部分程式決策模型

此階段由於超音波測距不穩定性較高，僅可確保越接近障礙物越精確，因此先行依測距判斷是否需微調角度，過於理想化。

B.2 接近障礙物時才做轉彎微調：

將理想斜邊值改為 20 公分，以避障方法找尋可走路徑。

a. 前方有三個障礙物

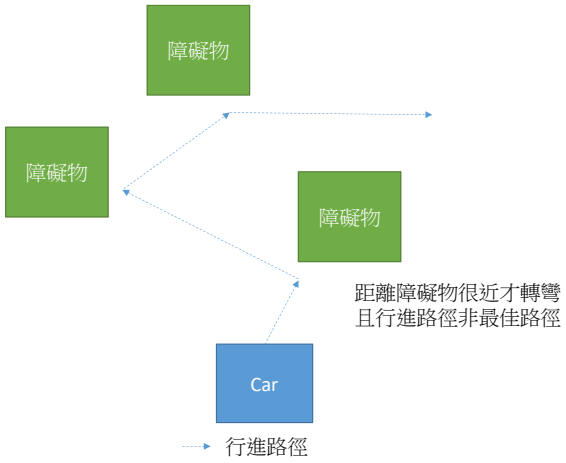
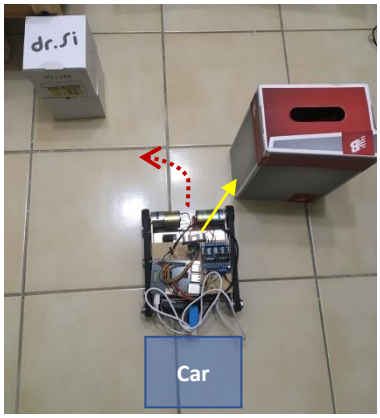
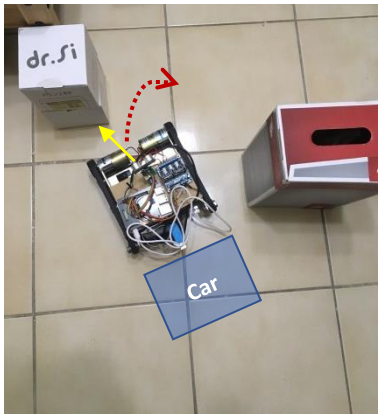


圖 12、障礙物與車子位置示意圖(二)



b.直行後，測出右側障礙物

c.左轉



d.直行後，測出左側障礙物

e.右轉

f.直行後，測出左側障礙物

g.右轉完畢，直走

圖 13、障礙物與車子實際位置圖

此階段由於車子的行進路徑會於離障礙物極短距離才轉彎，且因為用微調的方式決定旋轉角度，每微調一次便需要再對各角度進行測距，以確定微調過的角度是否能避開障礙物，使得避障的過程過於冗長。(如圖 12 所示)

C. 階段三：增加座標(x, y)、地圖及階段性虛擬點

此階段同階段二實現多角度測距(角度同階段二)，比階段二多增加記錄偵測到的障礙物座標，藉由座標找出可行路徑之中點，先使車子移動到中點線上之預設虛擬點，再穿越障礙物，且因為希望車子行進路徑接近最短路徑，若左、右前方測距小於 40 公分，及判定有障礙物並計算其中點位置。且車子前進距離為所偵測之多方向測距的最小值。並記錄所有測出之障礙物座標，繪製出地圖，此階段之行進路徑比起階段二更接近最短路徑(如圖 16 所示)。

決策模型：

Default：車子座標為(0, 0)

If (前距>60 公分)且(左右前方可行)：

前進前距之一半

更新車子的 y 座標

預設車子起始座標(carX, carY)為(0, 0)
若前測距大於 60 公分且左、右前方無障礙物

else if (前距<=60 公分)或(左右前方不可行)：

min = 各角度測距之最小值

找出前方空隙(goalX, goalY) //虛擬點

求出車子與虛擬點座標差距(absX, absY)

Degree = $\tan^{-1}\frac{absX}{absY}$

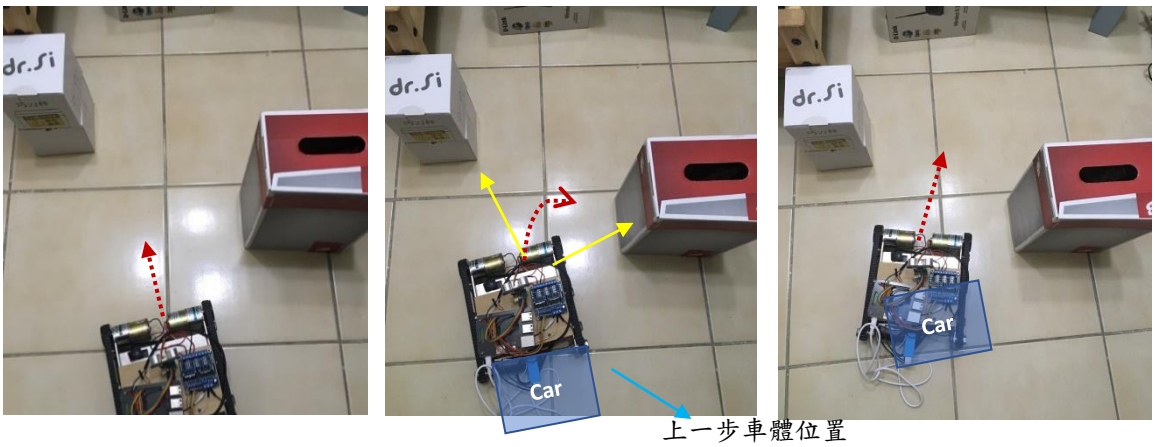
求車子與虛擬點之距=$\sqrt{absX^2 + absY^2}$

往虛擬點移動並更新 x 座標及 y 座標

若前測距小於 60 或左、右前方有障礙物，先設定相對車子之虛擬點(goalX, goalY)，再算出絕對值(absX, absY)，依畢氏定理與三角函數求得車子與虛擬點之夾角與距離。

更新車子座標(carX, carY)
設車子與虛擬點夾角為 *degree*
carY = 斜邊 × $cos(degree)$
carX = 斜邊 × $sin(degree)$

圖 14、階段三部分程式決策模型



a.前方有三個障礙物　b.直行後，測出空隙位置　c.轉彎

d.直行後，測出右前方空隙位置　　e.轉彎　　f.直行離開

圖 15、障礙物與車子實際位置圖

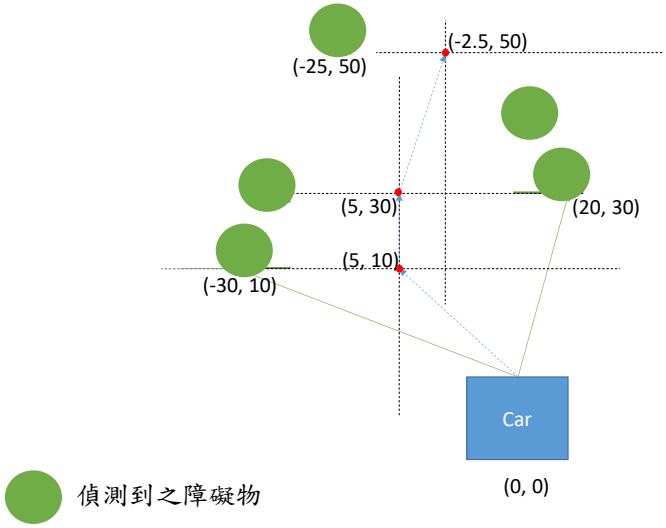


圖 16、地圖繪製

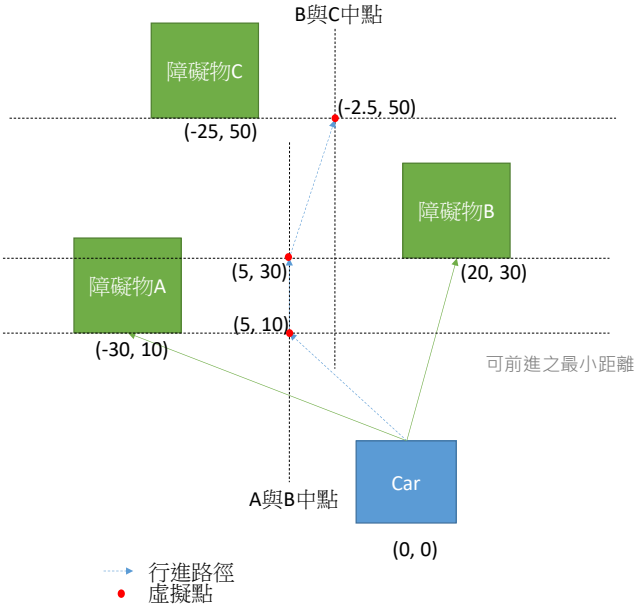


圖 17、障礙物與車子位置示意圖(三)

經由三階段實驗，將探測車之超音波避障演算法完成至適合之決策模型，並藉由座標化之概念將障礙物絕對位置圖繪製出來，點越密集處代表該位置障礙物可能較大，但由於超音波測距對於距離較遠之物體準確度較差，可能無法準確測出較遠處是否有障礙物，因此容易又走向近距離避障之演算法，為求尋找最佳路徑、高效率，我們將結合影像辨識方法，先辨識出障礙物位置，再由超音波測距。

第四章　影像辨識

4.1 Convolution Neural Network

CNN 主要分為三部分，依序為卷積層(Convolutional layer)、池化層(Pooling layer)、全連接層(Full connected layer)，與傳統類神經網路不同的是 CNN 增加卷積層及池化層。傳統類神經網路須輸入一維資料，但影像包含水平像素、垂直像素、色彩(color channel)三維資料，將其轉為一維資料會失去重要空間資料，不同影像但類似空間可能有相似像素值，RGB 不同的 channel 間也可能具某些關連性，而遠近不同的像素彼此也應具不同關聯性，這些資訊只在三維形狀中才能保留。

卷積層的原理是透過一個固定大小的視窗(Convolution kernel)，由上而下依序滑動取得圖像中各局部特徵作為池化層的輸入，池化層將輸入的圖片尺寸縮小以減少每張特徵圖的維度並保留重要的特徵，經過數次的卷積及池化，將結果傳至全連接層進行分類，相較於卷積與池化全連接層的成本是相當的高，主要的類神經網路集中於此層中。

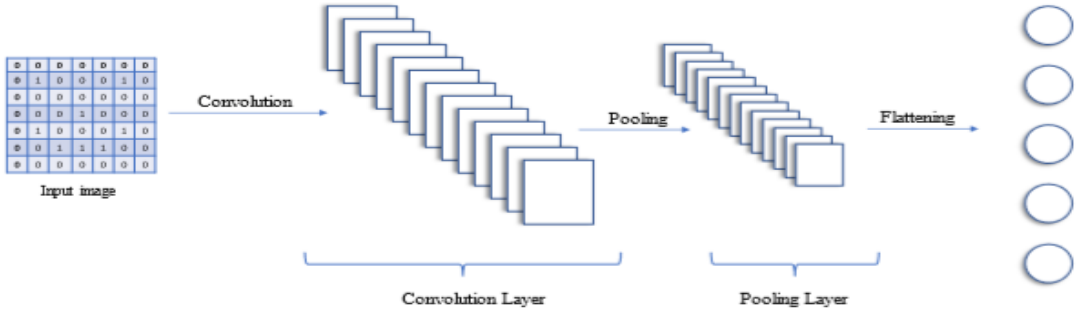


圖 18、CNN 概念圖

4.2 找出適合 Raspberry Pi 3 推論模型

CNN 為主要的類神經網路基礎架構，因此在 CNN 架構上衍生出不同的類神經網路模型，如 YOLO、Fast R-CNN、MobileNet 等等，由於需在 Raspberry Pi 3 中執行，從眾多模型中挑選辨識速度較為快速的模型比較，分別為 YOLOV2-tiny 及 SSD_MobileNet，兩模組均使用 coco dataset 分類數均為 90，兩模組在 Raspberry Pi 3 執行及時辨識的比較(如圖 19 所示)。

通過表格可得知 SSD_mobileNet 各分面表現均優於 YOLOV2-tiny，mAP 看出 SSD_mobileNet 較 YOLOV2-tiny 準確，在進行即時辨識時 YOLOV2-tiny 無法辨識 FPS 為 0，證明 YOLOV2-tiny 的運算量對 Raspberry Pi 3 的負擔過重，而 SSD_MobileNet 進行即時辨識最低為 3 秒一張。

模型	mAP	FPS
YOLOV2-tiny	-	無法辨識
SSD_mobileNet	19.3	0.353

圖 19、YOLOV2-tiny 與 SSD_mobilenet 比較

4.3 分析推論模型的時間消耗

推論模型中可分五大部分，載入各資料時間(loading time)、圖片前處理時間(convert time)、物件辨識時間(detection time)、產生結果輸出時間(print time)、視覺化顯示時間(show time)，於 Raspberry Pi 3 執行即時辨識並取前二十秒分析個別時間消耗(如圖 20 及圖 21 所示)。

	第一張	第二張	第三張	第四張	第五張	第六張	第七張	第八張	平均(忽略第一張)
載入各資料時間 (loading time)									51.157
convert time (圖轉numpy array時間)	0.002	0.001	0.002	0.001	0.003	0.002	0.003	0.002	0.002
detection time (tensorflow 偵測時間)	26.266	3.221	3.224	3.23	3.187	3.185	3.197	3.276	3.217142857
print time (輸出時間)	0.003	0.002	0.002	0.002	0.003	0.002	0.002	0.002	0.002142857
show time (圖片顯示時間)	0.124	0.045	0.043	0.043	0.043	0.042	0.043	0.043	0.043142857

圖 20、SSD_MobileNet 推論效能評估

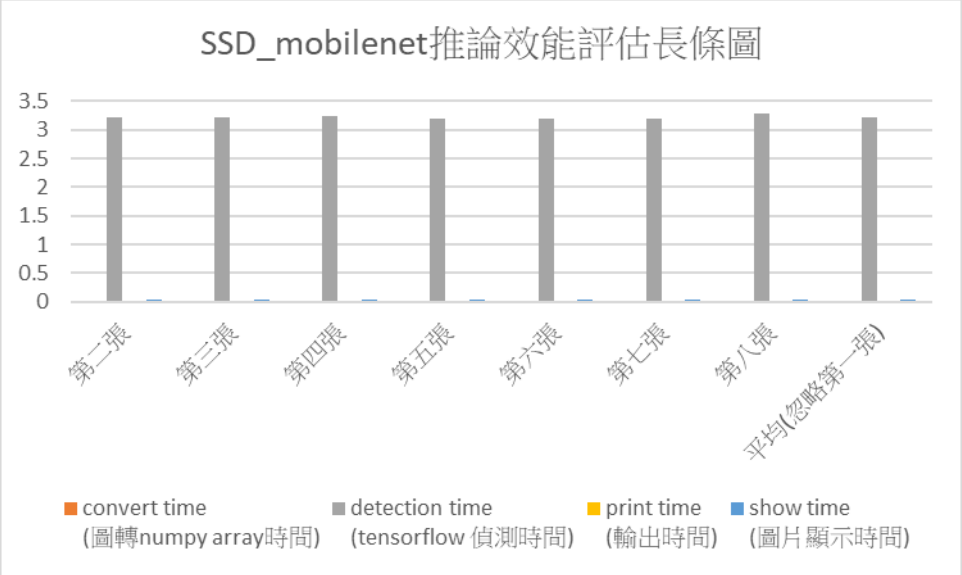


圖 21、SSD_MobileNet 推論效能評估長條圖

分析後得知第一張圖偵測時間較久，原因為第一次載入類神經網路需進行建置，因此後續平均時間將其忽略。通過平均得知時間消耗中載入各資料時間最久，而因資料執行程式時僅於一開始載入一次，不影響辨識時間，因此載入各資料時間將不考量。其次物件辨識時間於整體中占 98.56%，發現執行時的效能瓶頸在物件辨識，因此將針對 SSD_MobileNet 網路進行效能改善。

4.4 降低 SSD_MobileNet 分類數

得知效能瓶頸後，先嘗試降低 SSD_MobileNet 的分類數(90 類改為 1 類)，訓練完成後在 Raspberry Pi 3 進行推論，得到降低分類數的影響(如圖 22 所示)。

表格中可看出載入各資料時間及物件辨識時間有明顯降低(如圖 23 所示)，但物件辨識時間仍不如預期，無法實現及時辨識。

	第一張	第二張	第三張	第四張	第五張	第六張	第七張	第八張	第九張	第十張	平均(忽略第一張)
載入各資料時間 (loading time)											10.004
convert time (圖轉numpy array時間)	0.003	0.002	0.002	0.002	0.002	0.003	0.003	0.002	0.002	0.002	0.002222222
detection time (tensorflow 偵測時間)	10.073	2.334	2.307	2.332	2.353	2.326	2.362	2.341	2.34	2.335	2.336666667
print time (輸出時間)	0.002	0.002	0.008	0.002	0.002	0.003	0.003	0.003	0.004	0.004	0.003444444
show time (圖片顯示時間)	0.055	0.014	0.03	0.014	0.014	0.014	0.081	0.042	0.041	0.042	0.032444444

圖 22、降低分類數對時間的影響

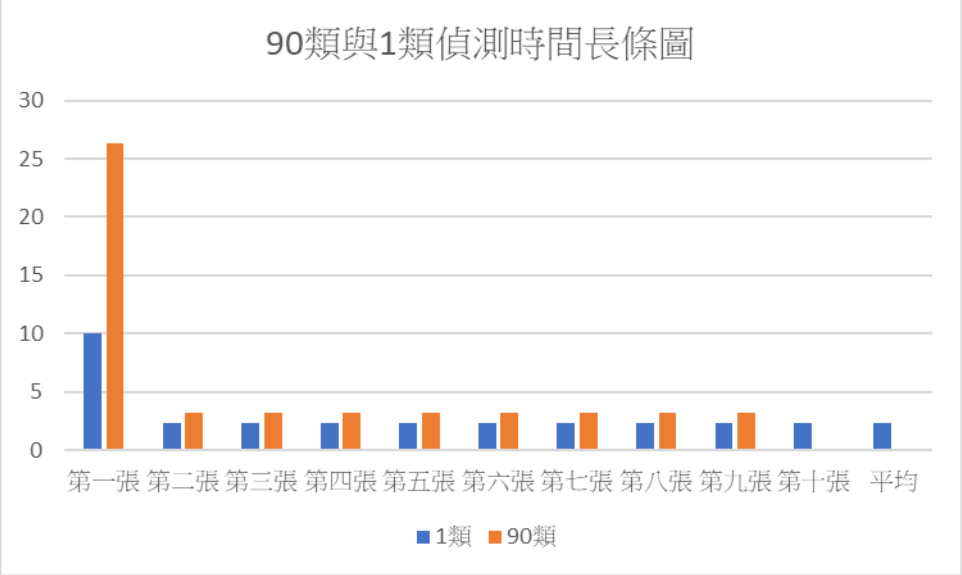


圖 23、分類數 90 類與 1 類偵測時間長條圖

因效能瓶頸依舊為物件偵測時間，決定對類神經網路本體進行修改並降低整體網路所需的運算量。

4.5 分析 SSD_MobileNet 模型

根據 SSD_MobileNet 原始訓練程式碼發現 SSD_MobileNet 是由 SSD(Single Shot MultiBox Detector)及 MobileNet 結合。通過原始論文與程式碼可得 MobileNet 僅是架構中一部分，並非使用完整 MobileNet，而 SSD_MobileNet 主架構為 SSD。

SSD 論文中提出 SSD 以卷積取代傳統 CNN 全連接層，SSD 中不存在 CNN 中最耗時之全連接層，並透過將分類及權重加入類神經網路中，在卷積運算時加入權重及分類，完成卷積運算時也完成物件分類及定位，透過六次卷積運算來分類出影像中不同大小的物件，將這六層的辨識結果透過 non-Maximum suppression 來輸出分類結果(如圖 24 所示)。

MobileNet 主要工作為提供 SSD 卷積層，透過 MobileNet 獨有的特殊卷積運算，將原本 CNN 中卷積的 RGB color channel 三維運算拆解為三個二維運算，原

本為 3*3*3 對 3*3*3 的卷積運算縮減為三個 3*3*1 對 3*3*1 的卷積運算，整體運算成本降低了 40%，加速了卷積層的運算。

SSD_MobileNet 的架構為使用 MobileNet 中的卷積層進行影像的卷積運算，將卷積運算結果給SSD 中取代全連接層六次卷積運算，透過此運算將結果輸出。

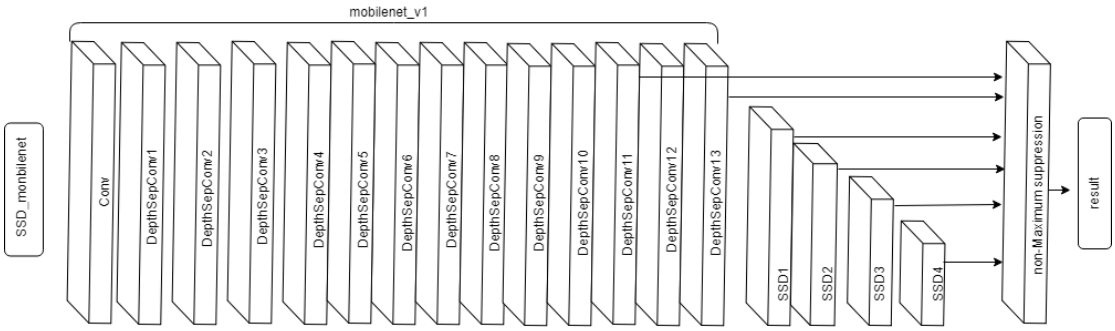


圖 24 SSD_mobilenet 架構圖

4.6 更改 SSD_MobileNet 模型參數

MobileNet 論文中提及，MobileNet 提供 depthwise 值做調整，可在精確度及效能上做不同效果，而 depth multiplier 是做卷積時僅保留一定比例 channel 再往下層卷積，可減少卷積運算量，原始 ssd_MobileNet 中 depth multiplier 值為 1，且程式碼中可看出 SSD_MobileNet 的 channel 皆為 8 的倍數，因此決定 depth multiplier 值以 0.5, 0.625, 0.75, 0.875,1 訓練並於 Raspberry Pi 3 上推論(如圖 25)。

depth multiplier	loss	mAP	FPS
0.875	1.424	14.6	0.380
0.75	1.396	15	0.429
0.625	1.430	13.9	0.517
0.5	1.503	12.3	0.582

圖 25 調整 depth multiplier 效能影響

a. Step 1092　　b. Step 30000

圖 26 depth multiplier0.5 訓練數據

根據表格(如圖 25 所示)可以看出在 depth multiplier0.5 loss 已經相當大 mAP 相對 0.5 以上的小，經過實測在 Raspberry Pi 3 雖然 FPS 較高但準確率相當低，因此決定不採用 depth multiplier 小於 0.5 的模型，且將 depth multiplier 大於等於 0.5 的模型其 SSD 部份後六層的卷積運算改為與 MobileNet 相同的 depthwise 運算方式，並且在 Raspberry Pi 3 上進行實測。

depth multiplier	loss	mAP	FPS
0.875	1.418	14.9	0.385
0.75	1.391	15.1	0.444
0.625	1.477	13.1	0.528
0.5	1.515	11.7	0.618

圖 27 採用 depthwise 的效能影響

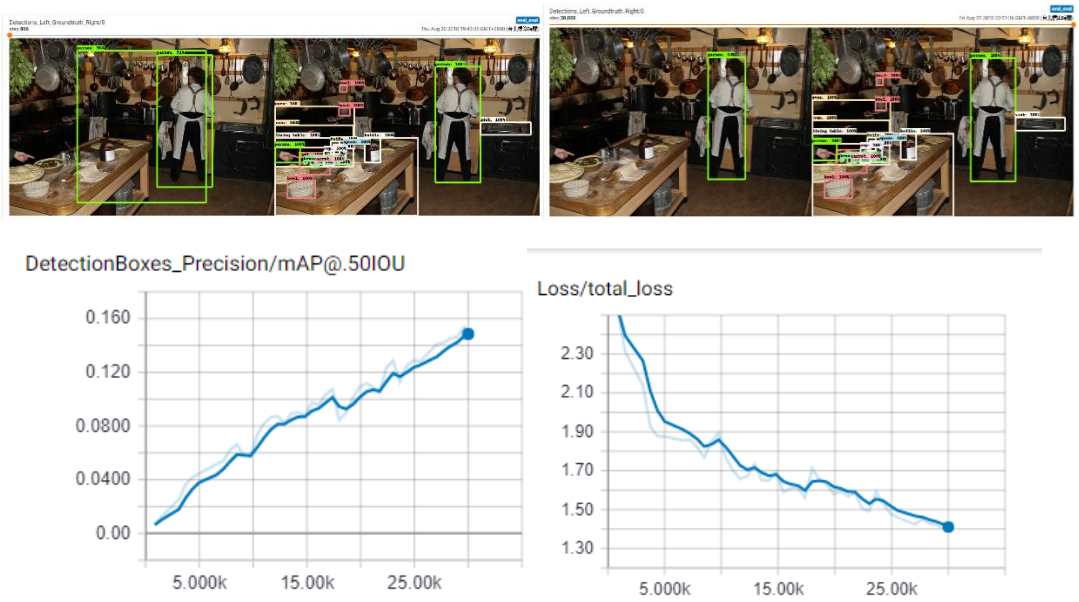


圖 28 depth multiplier0.75 & depthwise 訓練數據

表格中(如圖 27 所示)可發現將 SSD 後六層改用 depthwise 方式運算整體的 FPS 雖提高但不明顯，而 loss 及 mAP 卻也相對地提高及降低一些，當 depth multiplier 等於 0.75 時例外，不但 FPS 提高且 loss 降低 mAP 提高，此模型明顯優於其他模型，目前便以此模型為基準，並以縮小 SSD_MobileNet 網路拓樸為接續目標，持續實驗、分析。。

第五章　未來展望

本專題於 2018 年 7 月拿到車體零件後進行實際研究，此專題仍進行中，未來預計將第三章之避障加入 PID 控制模型，並與第四章之影像辨識結合，實現影像避障及影像搜索，另也將加入語音控制，使自走車執行命令，以下說明：

PID 控制模型部分將以目前階段三之避障數據，含轉彎角度與障礙物距離之關係、轉彎後之行走距離等數據，整理成表格，並利用 PID 之公式(如圖 29)計算出最佳轉彎角度與行走距離，期望使車子避障過程之行走路徑更為平順，以貼近實際最佳路徑為目標努力。除加入 PID 控制模型，也將使用影像辨識框出障礙物位置，並與超音波之測距進行交叉比對，判斷前方是否確實有障礙物，以排除誤判情況。

$R(t) = \text{set point} = \text{目標值}$
$Y(t) = \text{當前值}$
$E(t) = R(t) - Y(t) = \text{目標值} - \text{當前值}$
$P(t) = E(t) = \text{比例方程式}$
$I(t) = I(t-1) + I(t) = \text{積分方程式}$
$D(t) = E(t) - E(t-1) = \text{微分方程式}$
$U(t) = K_p \times P(t) + K_i \times I(t) + K_d \times D(t)$
$= \text{被控制的數值，其中} K_p \text{、} K_i \text{、} K_d \text{為增益值}$

圖 29、PID 公式

另語音控制部分，將預設指令，如：前進、後退、停止等命令，以語音方式透過 Google 提供之語音辨識 API，將輸入語音由音訊先轉為文字，再判斷所輸命令為何並執行，以下說明：

步驟一：實作Android App

步驟二：串接Google 語音辨識API

第六章　結論

整體來說，我們預計探測車須包含並整合以下功能，啟動後開始避障，並繪製災區地圖，而在避障同時，判斷Webcam拍攝的景象中是否有人臉，若偵測到人臉，探測車應往目標物前進。若接收到語音命令，應以執行語音命令為優先。

本專題探測車主要的目標為代替救災人員初步探索現場，使救災人員在真正進入災區前對現場有初步認知，也能事先知道危險程度，若在初步搜索過程中偵測到人，救災人員可立即往該處移動進行救援任務，也能藉由探測車提供的資訊，決定該帶何種器具進入災區、避免經過哪些危險區域，並及時救援已找到的生還者，如此減低人員危險度的同時，也延長了救援的黃金時間。

參考文獻

[1] Raspberry Pi 3 官網。取自 https://www.raspberrypi.org/。(Retrieved on Sep 19, 2018)

[2] TensorFlow官網。取自https://www.tensorflow.org/。(Retrieved on Sep 19, 2018)

[3] 盧嘉泓、林能祺(2009-04-21)。超音波避障裝置之設計。(Retrieved on Aug 19, 王信傑(2017-11-18)。當 PYTHON 遇上 RASPBERRY PI。取自 https://blog.everlearn.tw/%E7%95%B6-python-%E9%81%87%E4%B8%8A-raspberry-pi/raspberry-pi-3-mobel-3-%E5%88%A9%E7%94%A8-pwm-%E6%8E%A7%E5%88%B6%E4%BC%BA%E6%9C%8D%E9%A6%AC%E9%81%94 。(Retrieved on Aug 19, 2018)。

[4] 洪文麟(民105年7月)。深度學習應用於以影像辨識為基礎的個人化推薦系統-以服裝樣式為例

[5] 李文猶、黃峻昱、羅名哲、田懋祥、簡祥宸、楊士賢。救災機器人。

[6] Katina Michael,M. G. Michael.(June 26, 2014)。The Packbots Are Coming: Boosting security at the 2014 FIFA World Cup.。Published in IEEE Consumer Electronics Magazine

[7] Allen Taylor(May 9, 2015)。Meet SAFFiR: The Navy's Autonomous Fire Fighting Robot。取自http://www.topsecretwriters.com/2015/05/meet-saffir-the-navys-autonomous-fire-fighting-robot/。(Retrieved on Jan 31, 2018)

[8] 馬克周(2011)。人臉偵測Face Detection 使用OpenCV 2.4.2。取自 http://mark-jo-prog.blogspot.tw/2012/08/face-detection-opencv-242.html。(Retrieved on Jan 31, 2018)

[9] 機器人科技網(2017-08-27)。兩個角度量乾坤：三角測量原理及其應用。(Retrieved on Sep 15, 2018)

[10] Virtualwiz(2016-05-05)。我對『PID演算法』的理解 —— 原理介紹。(Retrieved on Sep 15, 2018)

[11] 林大貴(2017)。TensorFlow + Keras 深度學習人工智慧實務應用。博碩出版 。

[12] ●Wolfgang Beyer(Dec5, 2016)。How to Build a Simple Image Recognition System with TensorFlow (Part 1)。取自 http://www.wolfib.com/Image-Recognition-Intro-Part-1/。 (Retrieved on Jan 31, 2018)

●Wolfgang Beyer.(Dec31, 2016)。How to Build a Simple Image Recognition System with TensorFlow (Part 2)。取自 http://www.wolfib.com/Image-Recognition-Intro-Part-2/。(Retrieved on Jan 31, 2018)

[13] Wei Liu , Dragomir Anguelov , Dumitru Erhan , Christian Szegedy , Scott Reed,

Cheng-Yang Fu , Alexander C. Berg。SSD: Single Shot MultiBox Detector。取自 https://www.cs.unc.edu/~wliu/papers/ssd.pdf。(Retrieved on Dec 28, 2015)

[14] Andrew G. Howard , Menglong Zhu , Bo Chen , Dmitry Kalenichenko , Weijun Wang , Tobias Weyand , Marco Andreetto , Hartwig Adam。MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications。取自 https://arxiv.org/pdf/1704.04861.pdf。(Retrieved on Apr 17, 2017)

[15] Huang J , Rathod V , Sun C , Zhu M , Korattikara A , Fathi A , Fischer I , Wojna Z ,Song Y , Guadarrama S , Murphy K , CVPR 2017。Speed/accuracy trade-offs for modern convolutional object detectors。取自 https://arxiv.org/pdf/1611.10012.pdf。(Retrieved on Apr 25, 2017)

[16] Jake Bouvrie。Notes on Convolutional Neural Networks。取自 http://cogprints.org/5869/1/cnn_tutorial.pdf。(Retrieved on Nov 22, 2006)

[17] Alex Krizhevsky , Ilya Sutskever , Geoffrey E. Hinton。ImageNet Classification with Deep Convolutional Neural Networks。取自 https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf。(Retrieved on Dec 3, 2012)

[18] Cecilia F. Silva , Clauirton A. Siebra。An investigation on the use of convolutional neural network for image classification in embedded systems。取自 https://ieeexplore.ieee.org/document/8285727/。(Retrieved on Nov 8, 2017)

[19] Michael G. Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun。DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car。取自 https://arxiv.org/pdf/1712.08644.pdf。(Retrieved on Jul 30, 2018)

[20] Xingzhou Zhang, Yifan Wang, Weisong Shi。pCAMP: Performance Comparison of Machine Learning Packages on the Edges。取自 https://www.usenix.org/system/files/conference/hotedge18/hotedge18-papers-zhang.pdf。(Retrieved on Jul 30, 2018)

[21] Xingzhou Zhang, Yifan Wang, Weisong Shi。pCAMP: Performance Comparison of Machine Learning Packages on the Edges。取自 https://www.usenix.org/system/files/conference/hotedge18/hotedge18-papers-zhang.pdf。(Retrieved on 2018)

[22] Navinkumar Patle, Shweta Dhake, Devayani Ausekar。Automatic Object Sorting using Deep Learning。取自 https://www.aaai.org/Papers/AAAI/2006/AAAI06-361.pdf。(Retrieved on 2006)