

國立虎尾科技大學電機工程系
專題製作報告

指導教授：丁振聲 教授

無線導航機器人
視覺控制研究

班 級： 四技電機四甲
參與成員： 林彥名 林祐生
徐仁威 賴侑誠

中華民國 一百零二年 十月 十一日

摘要

在影像設備價格近年來大幅降低，以及電腦科學的進步，以視覺為基礎的研究為近年來熱門的研究主題。利用電腦視覺的方法，對影像進行分析。而其好處，包括節省人力資源、降低成本和提供多樣化的服務。

本專題主要是應用 OpenCV 此開發函式庫來完成影像分析與處理，藉由處理過後的影像讓無線導航機器人達到尋找目標與避障功能。

以 OpenCV 函式庫作為影像處理技術，包括均值濾波、邊緣檢測、侵蝕及擴張等影像處理方法，找出路徑上之障礙物或環境之目標物。結合 MobileRobots 的 ARIA 函式庫，讓無線導航機器人能前往目標點且閃避路徑上之障礙物避免碰撞。以及對周圍環境尋找目標以達到目標追尋之功能。

目錄

	頁數
摘要	I
第 1 章 緒論	1
1.1 前言	1
1.2 研究動機與目的	1
1.3 專題報告架構	1
第 2 章 專題系統的實驗設備	2
2.1 無線導航機器人硬體系統	2
2.2 影像系統	2
2.3 無線傳輸系統	3
2.4 軟體系統	3
2.5 本章總結	3
第 3 章 無線導航機器人之影像處理與導航	4
3.1 RGB 色彩模型	4
3.2 二值影像處理	5
3.3 Mean Filter 均值濾波器	6
3.4 Sobel 邊緣檢測	8
3.4.1 Sobel 邊緣檢測的流程圖與結果	9
3.5 影像膨脹	10
3.6 影像侵蝕	10
3.7 影像處理流程	11
3.8 避障方式	14
3.9 目標追尋方式	16
3.10 避障專案介紹	18
3.10.1 前置處理	18
3.10.2 避障機器人動作主程式	18
3.10.3 副程式	20
3.11 目標追尋專案介紹	24
3.11.1 目標追尋機器人動作主程式	24
3.11.2 副程式	24
3.12 影像處理程式介紹	27
第 4 章 模擬與實現	35
4.1 無線導航機器人之模擬	35
4.2 整體實驗結果	36
4.2.1 閃避障礙物與前往路徑點結果	36
4.2.2 目標追尋結果	40
第 5 章 結論與未來展望	43

5.1 結論	43
5.2 未來展望	43
參考文獻與書籍	44

圖目錄

圖 2.1 無線導航機器人正面.....	2
圖 2.2 無線導航機器人側面.....	2
圖 2.3 無線攝影機.....	2
圖 2.4 無線路由.....	2
圖 2.5 無線網路基地台.....	3
圖 2.6 整體系統架構圖.....	3
圖 3.1 RGB 彩色模型示意圖.....	4
圖 3.2 無線攝影機擷取之原始圖片.....	5
圖 3.3 二值化後有效像素(白色)和無效像素(黑色)之圖片.....	5
圖 3.4 二值化流程圖.....	6
圖 3.5 原始像素值(左圖)，經過均值濾波器後像素值(右圖).....	6
圖 3.6 3x3 均值濾波器的權值模板.....	7
圖 3.7 經均值濾波器後之比較.....	7
圖 3.8 Sobel 邊緣檢測流程圖.....	9
圖 3.9 Sobel 邊緣檢測後之比較.....	9
圖 3.10 膨脹處理範例.....	10
圖 3.11 侵蝕處理範例.....	10
圖 3.12 影像處理基本流程圖.....	11
圖 3.13 演算法步驟顯示結果.....	13
圖 3.14 導航流程圖.....	14
圖 3.15 補正角度說明.....	15
圖 3.16 菱形避障.....	15
圖 3.17 目標追尋流程圖.....	16
圖 3.18 額外搜尋動作.....	17
圖 3.19 與機器人建立遠端連結畫面.....	18
圖 3.20 進入安全距離顯示畫面.....	21
圖 3.21 進入避障動作顯示畫面.....	22
圖 3.22 當前角度與累計面積畫面.....	24
圖 3.23 發現目標物之畫面.....	25
圖 4.1 國立虎尾科技大學電機系二樓左側.....	35
圖 4.2 抵達起點 A.....	35
圖 4.3 抵達目標點 B 前往終點 C.....	36
圖 4.4 抵達終點 C 並停止.....	36
圖 4.5 室內閃避路徑上障礙物程序圖與影像處理圖.....	38
圖 4.6 以原規劃路徑前往終點 C 程序圖.....	39
圖 4.7 目標物於 1.5 公尺內目標追尋程序與影像處理圖.....	41
圖 4.8 目標物於 1.5 公尺至 2.3 公尺內目標追尋程序與影像處理圖.....	42

第 1 章緒論

1.1 前言

科技的進步與工業的發展，許多產業界皆以自動化為導向，自從工業機器人引入市場，目前在自動化製造已是不可或缺的生產工具。在現行工廠中，生產等過程皆需搬運的工作，若以無線導航機器人取代人為的操控，則可增加生產的效率、降低成本與風險。無線導航機器人無論是硬體或軟體技術研發在美國、日本與歐盟等國家已長達數十年之久，而無線導航機器人移動平台設計與製造，到目前為止其功能具有強大的擴充性、完整的 C++ API 程式介面、軟體函式庫一應俱全。無線導航機器人的應用也逐漸普及到居家清潔與居家保全之中，且相關的研究也持續進行，以期能使人類有更便利的生活。

1.2 研究動機與目的

隨著科技的進步，生活品質也提升不少，科技融入社會進入生活，讓我們生活越來越便利。本專題提出在室內環境中，無線導航機器人能利用影像判斷出路徑上障礙物，以避免碰撞繼續前往目標點。此方法可應用在較簡單的室內環境如：室內巡邏機器人，在巡邏過程中閃避環境中的障礙物，以完成室內巡邏工作。或是尋找環境中之目標物，前往該物所在位置。如：居家打掃機器人，完成打掃工作後，尋找起始位置回歸。所以加入影像系統後將使得機器人可以執行更多的動作，也將有更多的應用。

1.3 專題報告架構

本專題報告共分為 5 章，第 1 章為緒論，介紹目前發展現況與動機；第 2 章介紹使用到的軟、硬體架構和影像系統；第 3 章影像處理和導航，介紹基本影像處理、避障和目標追尋方式；第 4 章實驗結果，呈現測試結果；第 5 章為結論和未來展望，對本專題做總結，以及未來可以改善之處做分析檢討。

第 2 章 專題系統的實驗設備

本章將針對本專題之軟硬體系統架構做詳細且完整的介紹，其內容將以硬體系統、影像系統、無線傳輸系統、軟體系統介紹等細節加以說明。

以下圖為實際無線導航機器人系統：



圖 2.1 無線導航機器人正面



圖 2.2 無線導航機器人側面

2.1 無線導航機器人硬體規格

長/寬/高	33/28/15 cm
重量	3.6kg
聲納數	8 個
最大速度	75 cm/sec
最大旋轉速度	300 degrees/sec

2.2 影像系統

一台無線攝影機(型號:WS-CAMERA-USB)，採用 2.4GHz 無線射頻技術發送視頻及音頻訊號至接收器，在空曠空間使用傳輸距離可達 1.2Km 以上，可遠端遙控，內含照明及電池，攝影鏡頭拍攝角度寬廣，可取得高畫質影像。

一台無線路由(型號:WS-DDPHIPOWER)經由 USB 介面連結到個人電腦，有使用方便易於擴展、傳送距離遠、傳輸速度快、資料安全性佳、耗電量低、體積小等特性。我們最後接收到的圖像像素大小為 640x480 pixels。



圖 2.3 無線攝影機



圖 2.4 無線路由

2.3 無線傳輸系統

在無線導航機器人上有一個無線裝置，另外在個人電腦上也有一個無線裝置。在個人電腦上下達指令經由無線網路基地台(AP)來做傳送，無線網路基地台(AP)的標準請參閱 IEEE 802.11b 標準。所以我們使用無線網路基地台(AP)當作個人電腦傳輸指令給無線導航機器人的中繼站。



圖 2.5 無線網路基地台

2.4 軟體系統

在個人電腦操控系統是 Windows 7 和用 Microsoft Visual Studio 2008 來做為本專題發展工具。

使用由 Activmedia Robotics 無線機器人系列提供的兩個軟體分別為 Mapper3、MobileSim 分別來做規劃場地地圖、以模擬器代替機器人進行驗證及實驗機器人行走時的狀況。

使用 ARIA 由 Activmedia Robotics 無線機器人系列提供的物件導向應用程式介面，ARIA 是用 C++ 語言撰寫，所以我們使用 ARIA 來做控制及取回無線導航機器人的狀態。

使用 OpenCV 由 Intel 公司所開法的跨平台計算機視覺函式庫，本次專題將 OpenCV 安裝在 Microsoft Visual Studio 2008 中，OpenCV 是用 C++ 語言撰寫，來做為影像處理之用。

2.5 本章總結

本專題之研究重點為利用無線導航機器人結合影像處理來達成避障與目標追尋的功能。所以使用無線攝影機及其他設備裝置完整系統架構如圖 2.6 示。

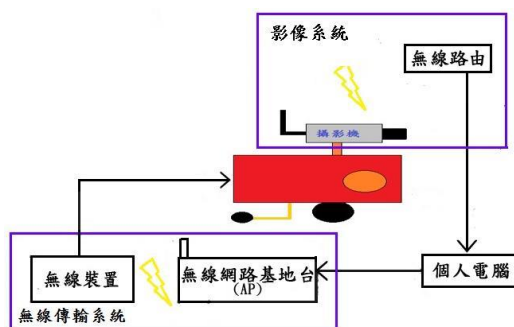


圖 2.6 整體系統架構圖

第 3 章 無線導航機器人之影像處理與導航

所謂影像處理，就是將擷取到的影像作數位化以利電腦運算。近年來，隨著技術的進步，影像處理已經被廣泛的運用在各個領域中，而且有相當的成就。像是醫學圖像分析、車牌辨識、智慧型機器人、安全辨識等都會用到機器視覺。在許多人類無法精確感知辨識的地方，如危險區域、宇宙影像等，更加可以顯示出機器視覺的優異性，且影像更可以讓人容易理解其內容。因此本專題使用無線攝影機擷取影像來分析處理，藉以模仿人類看到特定圖形理解圖形所代表的意義。

3.1 RGB 色彩模型

色彩模型的目的是便於以某種標準且一般可接受的方式來指定色彩。在 RGB 模型中，每種色彩是以紅、綠、藍的主要頻譜成分來顯現。此模型是建立在直角座標系統基礎上。其中 RGB 值是在三個頂點上；青色、紫紅和黃色是在另外三個頂點上；黑色在原點上、而白色是在離頂點最遠的頂點上。在此模型中，灰階、是從黑色到白色沿著連結這兩個點之間的線。此模型中的不同色彩是位於立方體上或其內部的點，且用從原點延伸出的向量來定義。用來表示在 RGB 空間每個像素所用的位元數稱為像素深度 (pixel depth)。考慮一個 RGB 影像，其中的紅色、綠色和藍色影像都是一個 8 位元的影像，在這種情況下，每個 RGB 彩色像素有 24 位元的深度。全彩(full-color)影像這個術語通常用來表示 24 位元的 RGB 彩色影像。圖 3.1(a)展示相對於圖 3.2(b)中之圖形的一個 24 位元的 RGB 彩色立方體。

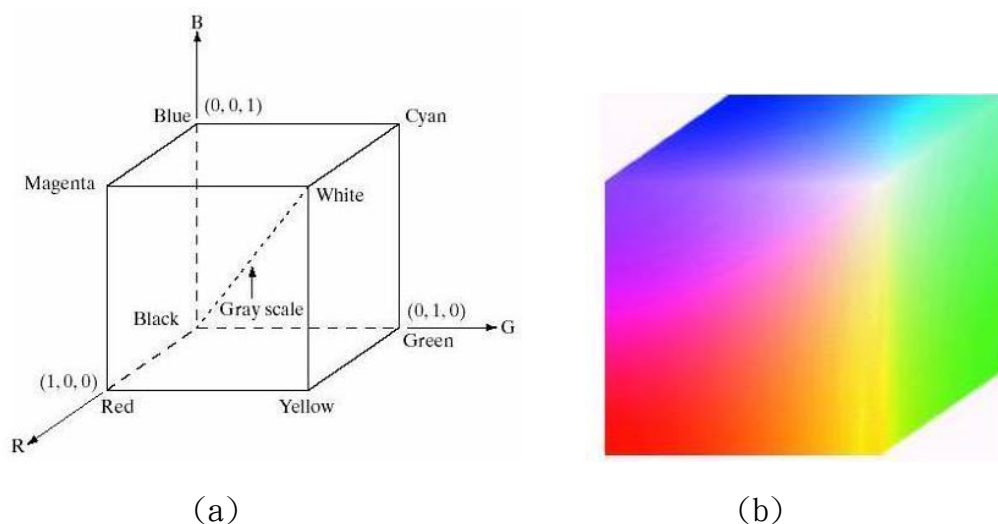


圖 3.1 RGB 彩色模型示意圖

3.2 二值影像處理

一般影像所需要的影像處理時間較長，但是成效不一定比較好(相較於二值化影像)。因此，大部分實用的機械影像視覺都是以二值化影像為主。且在影像處理技術當中，常常從一張複雜的原始圖中簡化成二值影像，方便後續處理，二值影像只會剩下黑色與白色的影像，也可以說留下有效像素與無效像素的影像。在 RGB 三原色中影像陣列 A 可以表示成如下：

$$A = \begin{bmatrix} (R_{11}, G_{11}, B_{11}) & \cdots & (R_{1N}, G_{1N}, B_{1N}) \\ \vdots & \ddots & \vdots \\ (R_{M1}, G_{M1}, B_{M1}) & \cdots & (R_{MN}, G_{MN}, B_{MN}) \end{bmatrix} \quad (3.1)$$

此影像共有 $M \times N$ 個點，而在本次專題我們所擷取的影像中， $M=640$ ， $N=480$ ，矩陣元素即為該點的 RGB 值，也是所謂的像素值。如(3.1)式因為使用矩陣來表示，由矩陣的行與列來表示一點的 (R, G, B) 座標值，也是像素值。在本次專題時作中因使用 RGB 彩色模型，所以以 8 個 bit 長度的影像作處理，每一點共有三個值每個像素值介於 0 到 255 之間。

得知影像 RGB 值後決定適當門檻值(Threshold)，將圖形二值化，本專題對影像所讀出的 RGB 值使用的門檻值寫為 (R_T, G_T, B_T) 。

$$(R_{MN}, G_{MN}, B_{MN}) = \begin{cases} (0,0,0), & (R_{MN}, G_{MN}, B_{MN}) \leq (R_T, G_T, B_T) \\ (255,255,255), & \text{else} \end{cases} \quad (3.2)$$

(3.2)式中將小於 (R_T, G_T, B_T) 之值變成黑色，其餘變成白色，本專題白色為有效像素，黑色為無效像素。



圖 3.2 無線攝影機擷取之原始圖片

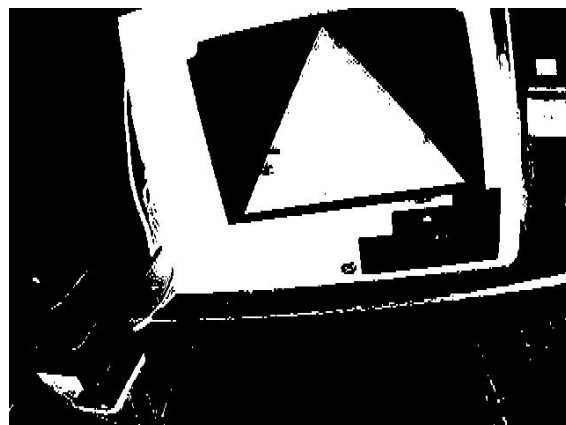


圖 3.3 二值化後有效像素(白色)和無效像素(黑色)之圖片

二值化的優點:1. 容易在實作中實現。2. 決定適當門檻值能夠當作分離物件的工作。以下為圖 3.4 二值化流程圖:

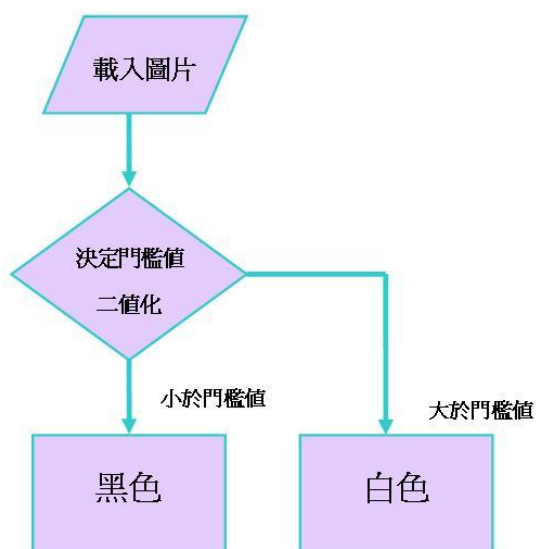


圖 3.4 二值化流程圖

3.3 Mean Filter 均值濾波器

濾波器其目的為去除雜訊，均值濾波器是一個線性平滑化濾波器。在一個二維空間中，均值濾波器的函數表示成:

$$h[i, j] = \frac{1}{S} \sum_{(x, y) \in R} f[x, y] \quad (3.3)$$

其中，S 是領域 R 內的像素點總數。例如：在像素點 [i, j] 處取 3x3 區域，得到：

$$h[i, j] = \frac{1}{9} \sum_{x=i-1}^{i+1} \sum_{y=j-1}^{j+1} f[x, y] \quad (3.4)$$

1	8	5
2	8	7
4	1	9

5	5	5
5	5	5
5	5	5

圖 3.5 原始像素值(左圖)，經過均值濾波器後像素值(右圖)

區塊 N 的大小控制著濾波程度，對應大模板的大尺度區塊會加大濾波程度。作為去除大雜訊的代價，大尺度濾波器也會導致圖像細節的損失，不同尺度的均值濾波器結果亦有所不同。

均值濾波器平滑化程度是由 S 來控制， S 值越大，平滑程度越高，相對的，影像越模糊。因此，我們可以容易的操控影像的平滑度。本專題使用的是 3×3 均值濾波器。

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

圖 3.6 3×3 均值濾波器的權值模板



(a.) 原始圖



(b.) 經灰階後的圖

(c.) 經濾波器後的圖

圖 3.7 經均值濾波器後之比較

從以上圖 3.7(b.) (c.) 我們可以看出影像變得較平滑。

3.4 Sobel 邊緣檢測

Sobel 是圖像處理中的算子之一，主要用作邊緣檢測。在技術上，它是一離散性差分算子，用來運算圖像亮度函數的梯度之近似值。在圖像的任何一點使用此算子，將會產生對應的梯度向量或是其法向量。

該算子包含兩組 3x3 的矩陣，分別為橫向及縱向，將之與圖像作平面卷積，即可分別得出橫向及縱向的亮度差分近似值。如果以 \mathbf{A} 代表原始圖像， \mathbf{G}_x 及 \mathbf{G}_y 分別代表經橫向及縱向邊緣檢測的圖像，其公式如下：

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \quad (3.5)$$

圖像的每一個像素的橫向及縱向梯度近似值可用以下的公式結合，來計算梯度的大小。

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2} \quad (3.6)$$

然後可用以下公式計算梯度方向。

$$\Theta = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right) \quad (3.7)$$

其中， θ 是梯度方向與+x軸的夾角。

-1	-2	-1
0	0	0
1	2	1

y方向的Sobel遮罩

-1	0	1
-2	0	2
-1	0	1

x方向的Sobel遮罩

3.4.1 Sobel 邊緣檢測的流程圖與結果

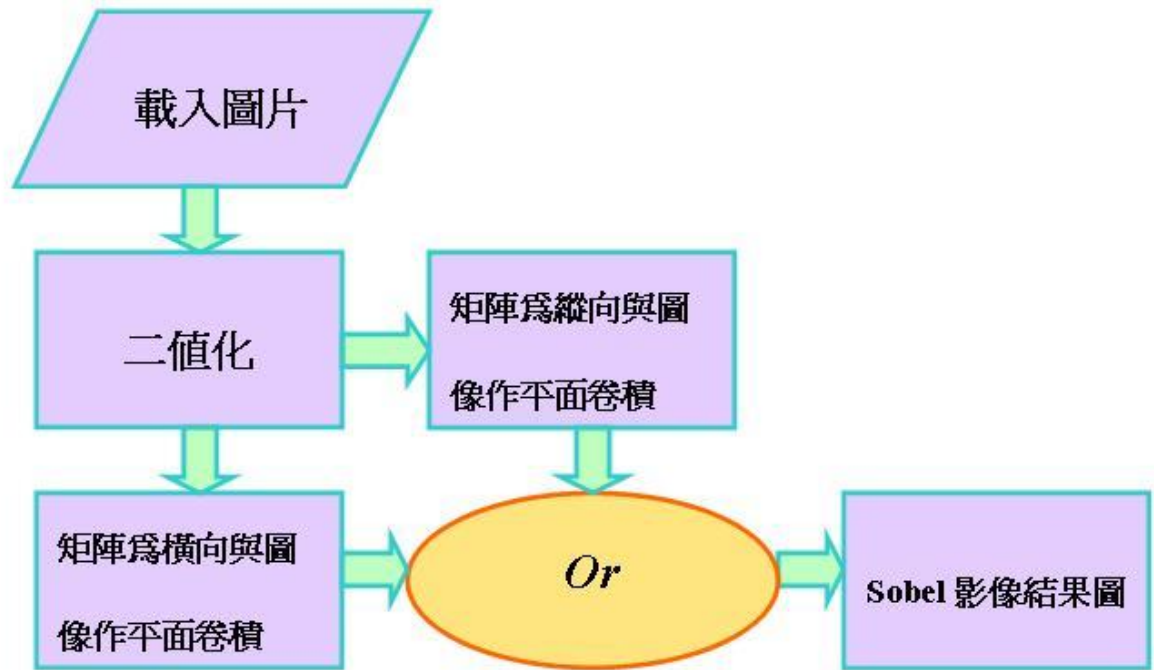


圖3.8 Sobel邊緣檢測流程圖



(a.) 原始圖

(b.)原圖經 Sobel 邊緣檢測後之結果

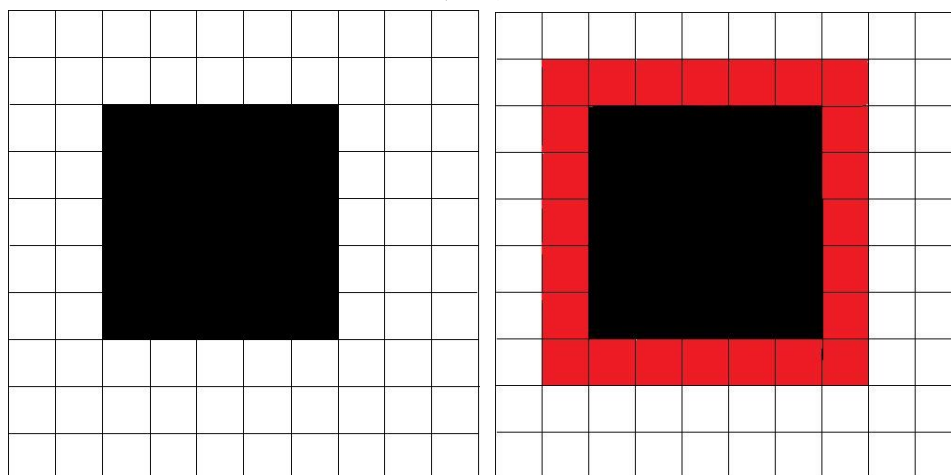
圖 3.9 Sobel 邊緣檢測後之比較

3.5 影像膨脹

膨脹是型態學影像處理中最基本的運算，其定義如下：

$$D(A,B) = A \oplus B = \bigcup_{b \in B} (A + b) \quad (3.8)$$

其中 B 是一個結構元素，而 A 是要被膨脹的集合(影像物件)。膨脹使二值化影像中的物件擴大，此擴大的特定形式和程度由所用的結構元素的形狀所控制。圖 3.11 膨脹處理範例。



(a.) 原始二值影像

(b.) 膨脹後結果

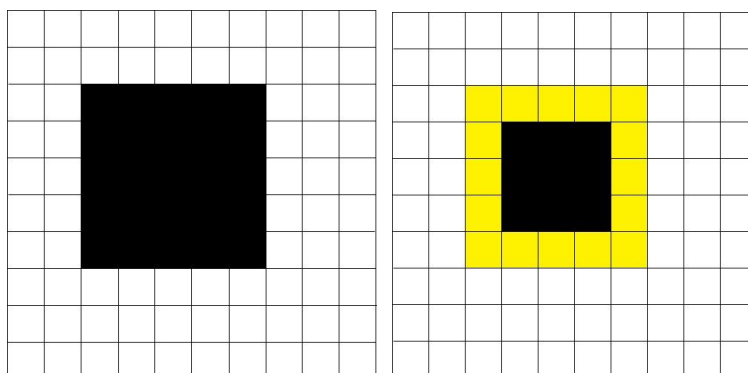
圖 3.10 膨脹處理範例

3.6 影像侵蝕

侵蝕也是型態學影像處理中最基本的運算，其定義如下：

$$E(A,B) = A \ominus (-B) = \bigcap_{b \in -B} (A - b) \quad (3.9)$$

其中 B 是一個結構元素，而 A 是要被侵蝕的集合(影像物件)。侵蝕使二值化影像中的物件縮小，此縮小的特定形式和程度由所用的結構元素的形狀所控制。



(a.) 原始二值影像

(b.) 侵蝕後結果

圖 3.11 侵蝕處理範例

3.7 影像處理流程

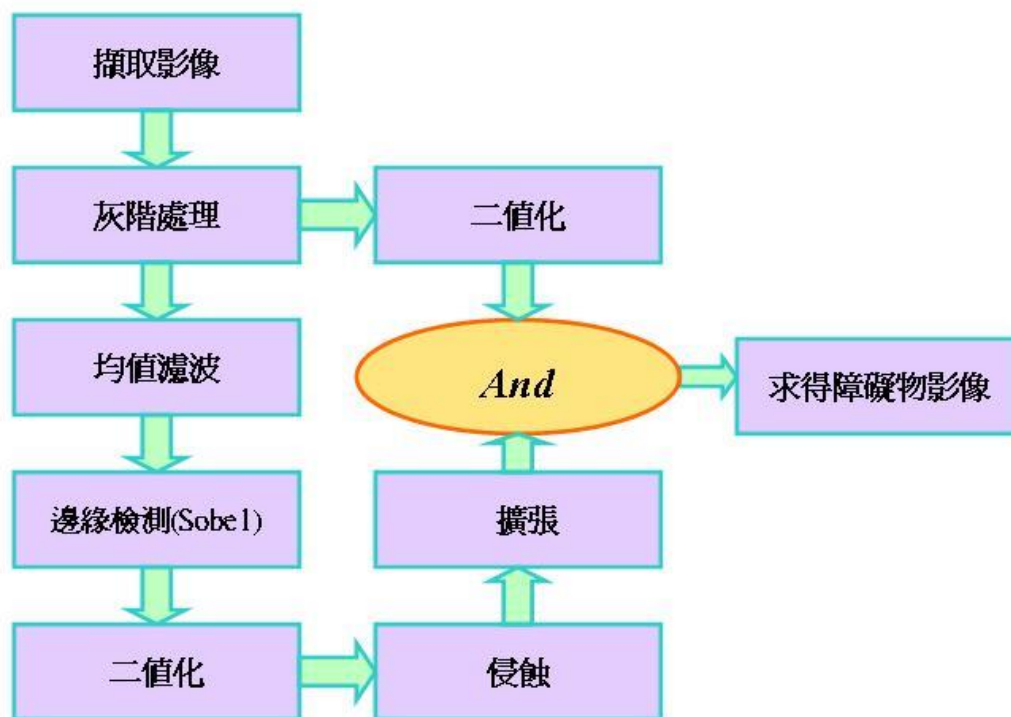
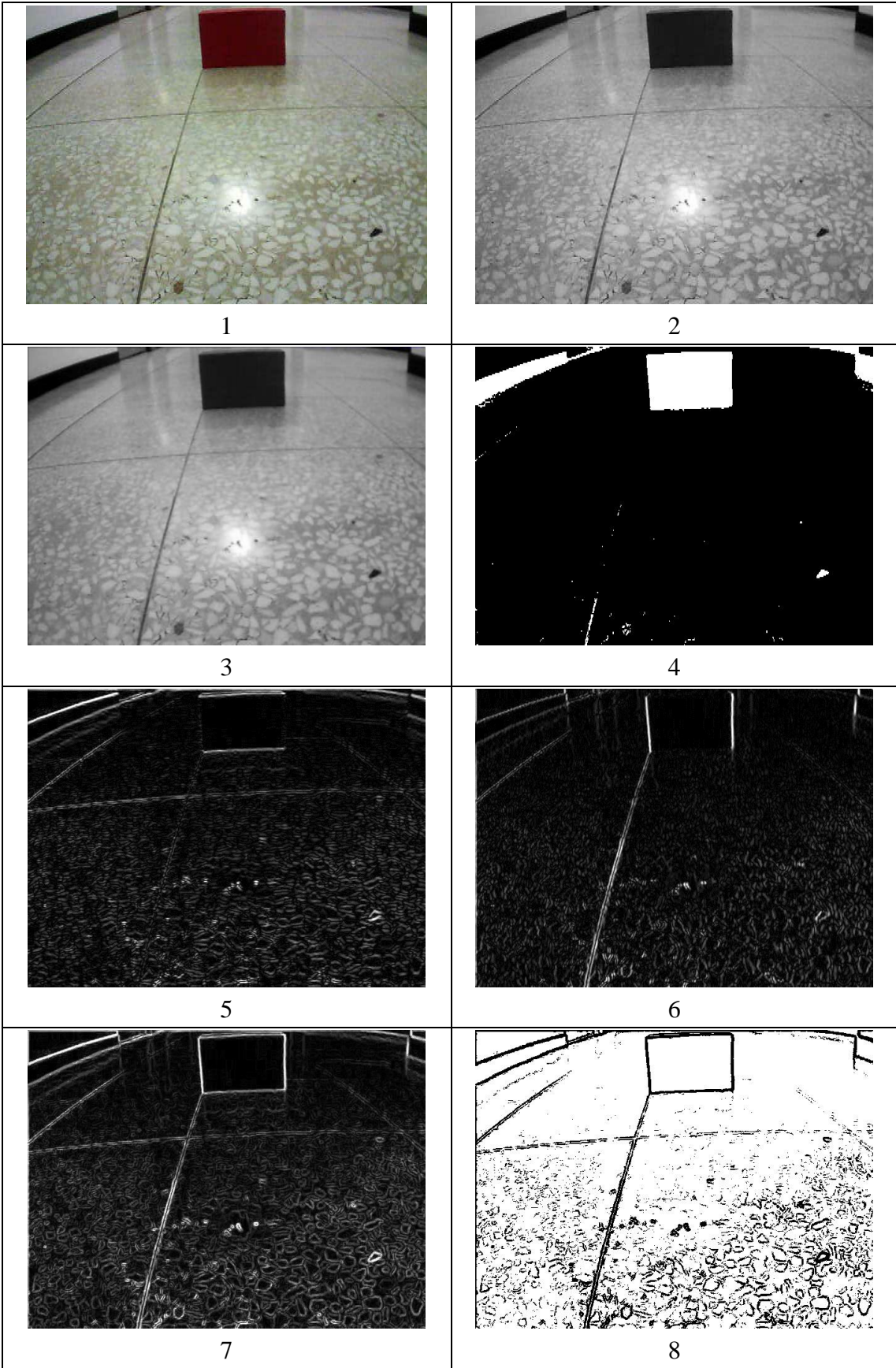


圖 3.12 影像處理基本流程圖

由於基本流程無法達成我們的需求所以我們增加了幾行步驟。

以下為演算步驟：

1. 擷取圖片(640*480)
2. 灰階處理
3. 將灰階處理後的圖做均值濾波
4. 另一張灰階處理後的圖做二值化等作比較
5. 濾波後做邊緣檢測(Sobel X 向量)
6. 濾波後做邊緣檢測(Sobel Y 向量)
7. 邊緣檢測(Sobel X 向量 Y 向量之合)
8. 邊緣後二值化
9. 侵蝕
10. 擴張
11. 比較且合併
12. 濾除不連續點(雜質)
13. 再做一次侵蝕
14. 再做一次擴張
15. 標記座標點
16. 標出障礙物



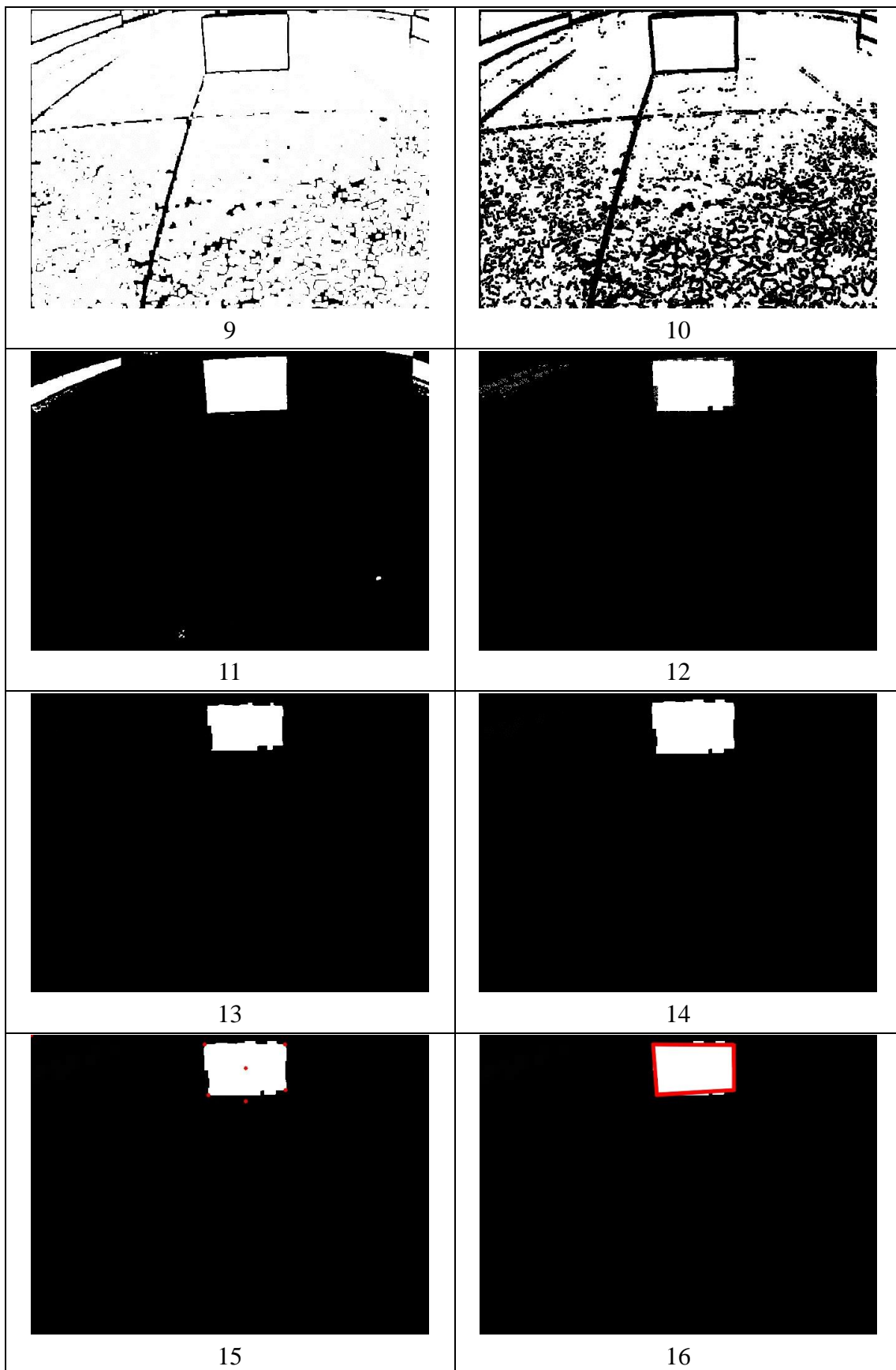


圖 3.13 演算法步驟顯示結果

3.8 避障方式

我們設定機器人前往目標點時，攝影機以每秒鐘擷取圖片一次，進行影像處理。藉由處理過後的影像，了解機器人前方之環境是否有威脅機器人行走之障礙物。如果發現障礙物，利用影像給予的資訊進行運算，算出目前障礙物與機器人之直線距離、障礙物與左右邊界的距離以及避開障礙物之左右補正角度。有了這些數值，我們就能讓機器人進行菱形避障。

菱形避障，顧名思義，以走菱形路徑來避開障礙物。首先，必須知道障礙物偏左或偏右，由此判斷機器人的左右兩側哪邊比較寬以方便進行避障動作。之後，讓機器人轉向較寬那邊的補正角度後，行走障礙物與機器人之直線距離再轉向較窄之補正角度後再次行走障礙物與機器人距離，完成避障動作，讓機器人繼續前往目標點。

此避障動作之優點為簡單、容易且具有回歸路徑之功能。缺點為障礙物不能過偏左右兩側，否則避障後容易會有偏離路徑之問題。

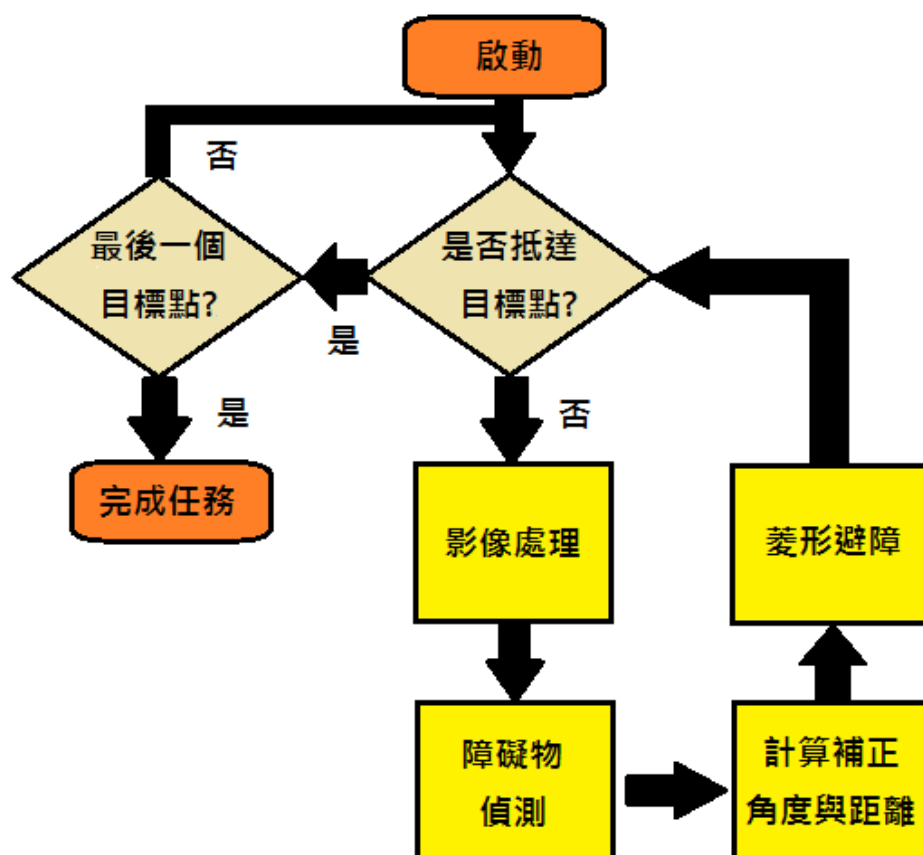


圖 3.14 導航流程圖

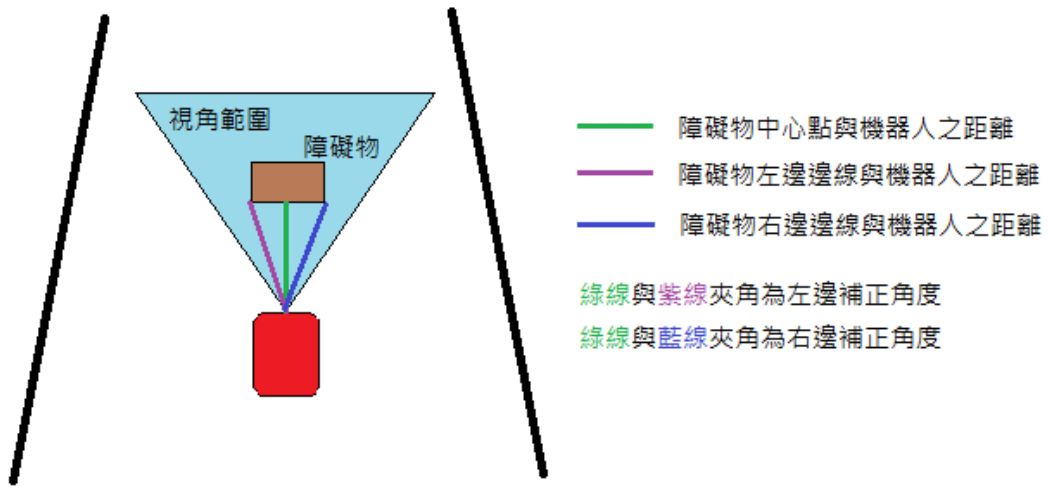


圖 3.15 補正角度說明

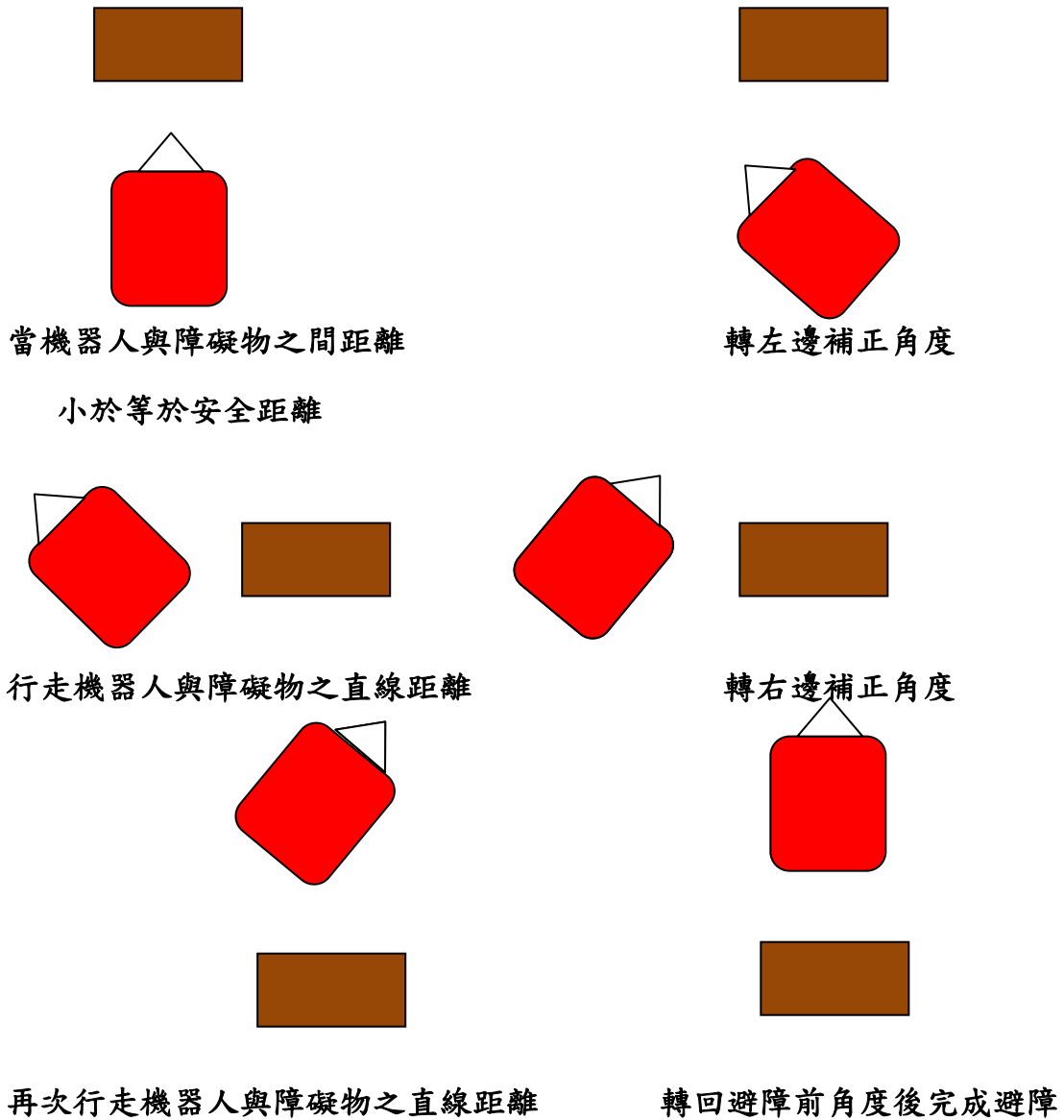


圖 3.16 菱形避障

3.9 目標追尋方式

機器人到達指定位置後，原地旋轉以尋找目標物，以每旋轉 10 度拍攝圖片做影像處理並依照機器人角度分為右側、左側紀錄面積。如機器人原地旋轉一圈，尚未發現目標物，將利用此為依準判斷目標物可能之方向，前往該方向進行額外搜尋動作。

當影像處理且判斷出目標物後，其資訊可讓機器人算出與目標物之距離、與目標物之偏離角度、機器人應抵達之位置，進行目標追尋動作。轉向偏離角度，前往抵達點之換算距離，完成動作。

額外搜尋動作。當機器人完成原地旋轉且未發現目標物將進入該動作，機器人比較左右面積紀錄值，向較大值之方向前進 60cm，再進入半圈搜尋動作，直到尋找到目標物。

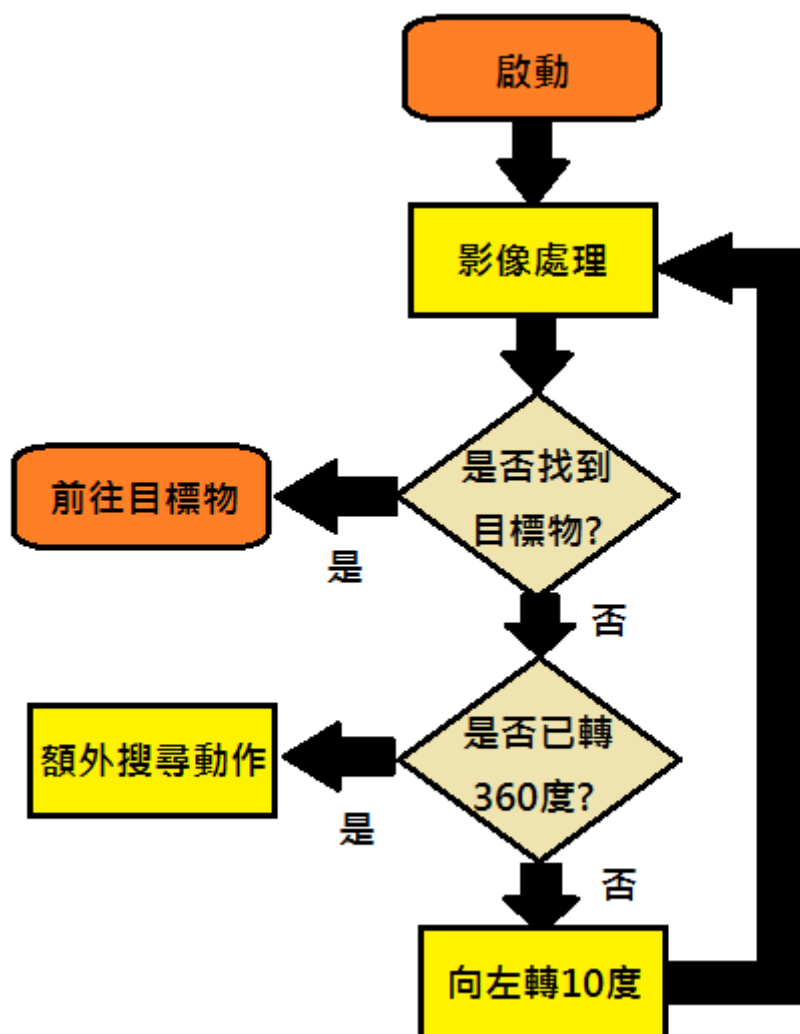
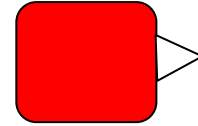
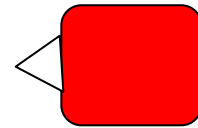


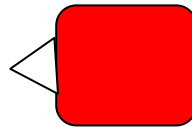
圖 3.17 目標追尋流程圖



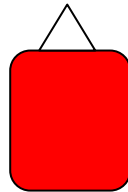
當完成全角度搜尋後，無判斷出目標物，但疑似有目標物存在。比較左右側紀錄之面積後，左邊面積較大，將對左邊進行額外搜尋動作



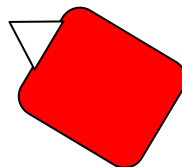
轉向左邊



前進 60cm



只對左半邊進行搜尋



發現目標物，前往目標物，結束目標追尋

圖 3.18 額外搜尋動作

3.10 避障專案介紹

3.10.1 前置處理

(1)ARIA 前置宣告

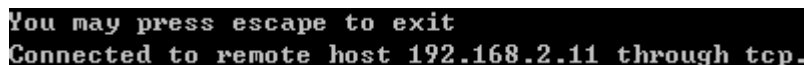
```
Aria::init(); //初始化ARIA函式庫
ArArgumentParser parser(&argc, argv); //啟動參數解析器
parser.loadDefaultArguments(); //載入預設參數
ArSimpleConnector simpleConnector(&parser); //建立機器人連結器，並傳入參數解析器
ArSonarDevice sonar; //聲納物件
ArAnalogGyro gyro(&robot); //機器人物件位址加入誤差調整器
robot.addRangeDevice(&sonar); //宣告偵測範圍裝置為聲納物件
ArKeyHandler keyHandler; //鍵盤操控器物件
Aria::setKeyHandler(&keyHandler); //使ARIA能使用鍵盤物件，預設值已能使用Esc鍵來退出程序
robot.attachKeyHandler(&keyHandler); //將鍵盤操控器加入機器人，每個工作週期皆會檢查按鍵
printf("You may press escape to exit\n");
```

```
if (!Aria::parseArgs() && !parser.checkHelpAndWarnUnparsed())
//若程式發現參數無法解析即送出關閉指令
{
    Aria::logOptions();
    exit(1);
}
```

(2)建立機器人連線

```
if (!simpleConnector.connectRobot(&robot))
//與機器人建立連結，若無法建立建立，立即送出關閉指令
{
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
}
robot.runAsync(true); //以非同步模式運行
robot.enableMotors(); //啟動機器人馬達
robot.comInt(ArCommands::SOUNDTOG, 0); //傳送啟動聲音指令到機器人物件上
```

執行畫面：



```
You may press escape to exit
Connected to remote host 192.168.2.11 through tcp.
```

圖 3.19 與機器人建立遠端連結畫面

3.10.2 避障機器人動作主程式

```
while (Aria::getRunning()) //只要跟機器人處於連接狀態，Aria::getRunning()會持續回傳True
{
    movePos(1000, 0, 5000); //移動至座標點(0, 1000)，等待動作完成時間為ms
    turn(nowangle, 5000); //讓機器人旋轉至絕對角度，大小值為nowangle(前面宣告值為-90)，等待動作完成時間為ms
    printf("抵達起點A，目前座標(0, 1000)\n");
    printf("前往目標點B\n");
    robot.setVel12(100, 100); //設定兩輪速度都為mm/s
    takepicture(); //呼叫攝影機拍照進行影像處理
    timer(1); //呼叫計時器
    while(startmoving==1)
    {
        converttotaldistance=converttotaldistance+totaldistance;
        printf("當前機器人座標位置( %d, 1000 )\n", converttotaldistance);
        if(pictimes==1) //當計時器完成時進入動作
        {
            takepicture();
            totaldistance=totaldistance+160;
            pictimes=0; //Reset其值
        }
        if(z==1)
        //判斷影像處理後是否看到障礙物
    }
}
```

```

{
    distance(mlx,mly); //計算障礙物與機器人之副程式
    Safedistance(80); //安全距離之副程式
    avoidobstacle(); //進行避障動作
    totaldistance=totaldistance+(2*(realobstacledistance+24)*10);
    z=0; //避障後，Reset判斷值
    robot.setVel2(100,100); //重新設定兩輪速度
}
if(pictimes==0) //重新呼叫計時器
{
    timer(1);
}
if((totaldistance>=2700))
//判斷機器人是否抵達目標點B
{
    startmoving=0;
    //給予判斷值，跳出迴圈。讓其進入下一迴圈
}
    converttotaldistance=constanttotaldistance;
}
turn(nowangle-90,8000);
nowangle=-180;
printf("抵達目標點B，目前座標(%d, 1000)\n",totaldistance);
printf("前往終點C\n");
takepicture(); //呼叫攝影機拍照進行影像處理
timer(1); //呼叫計時器
while(startmoving==0)
{
    converttotaldistanceB=converttotaldistanceB+totaldistanceB;
    printf("當前機器人座標位置(%d,%d)\n",
        totaldistance,1000-converttotaldistanceB);
    if(pictimes==1) //當計時器完成時進入動作
    {
        takepicture();
        totaldistanceB=totaldistanceB+160;
        pictimes=0; //Reset其值
    }
    if(z==1) //判斷影像處理後是否看到障礙物
    {
        distance(mlx,mly); //計算障礙物與機器人之副程式
        Safedistance(80); //安全距離之副程式
        avoidobstacle(); //進行避障動作
        totaldistanceB=totaldistanceB+(2*(realobstacledistance+8)*10);
        z=0; //避障後，Reset判斷值
        robot.setVel2(100,100); //重新設定兩輪速度
    }
    if(pictimes==0) //重新呼叫計時器
    {
        timer(1);
    }
    if(totaldistanceB>=2100) //判斷機器人是否抵達終點C
    {
        startmoving=-1; //給予判斷值，跳出迴圈
        printf("抵達終點C，座標(%d, %d)\n",
            totaldistance,1000-converttotaldistanceB);
    }
    converttotaldistanceB=constanttotaldistanceB;
}
Aria::shutdown(); //關閉ARIA所有的處理緒/執行緒並且退出程式
return 0;
}

```


3.10.3 副程式

(1) 旋轉角度

以開機當下角度為基準，使機器人旋轉一個角度。

```
void turn(double a,unsigned int b) //旋轉至絕對角度
{
    robot.setHeading(a); //旋轉至a絕對角度
    ArUtil::sleep(b); //等待動作完成時間，單位為ms
}
```

(2) 移動至座標

以開機位置為原點之相對座標點，單位為毫公尺(mm)。

```
void movePos(double a, double b, unsigned int c) //移動至座標點
{
    gotoPoseAction.setGoal(ArPose(a,b)); //移動至座標點(b,a)
    ArUtil::sleep(c); //等待動作完成時間，單位為ms
}
```

(3) 將影像點轉換成與機器人實際距離之副程式

給予影像座標點求出該點與機器人之實際距離，用於算出與障礙物距離與目標追尋距離。

```
void distance(int Px,int Py) //將影像點轉換成與機器人實際距離之副程式
{
    printf("投影座標點(%d , %d)\n",int (Px), int (Py));

    Px = Px-320; //將影像點轉換成
    Py = 480-Py; //相對於機器人之座標點
    printf("以機器人為中心點之投影座標點(%d , %d)\n",int (Px), int (Py));

    Y = Yh*tan(((90-alpha)+(Py/Sy)*(alpha-theta))*M_PI/180);
    //利用轉換計算，算出以機器人為中心點之實際Y軸距離
    if(Px == 0)
        beta = 0; //Px不得為零，否則Beta為無解
    else
        beta = (theta*Sx)/Px;
    X = Y*(Px/Sx)*tan(beta*M_PI/180);
    //利用轉換計算，算出以機器人為中心點之實際X軸距離
    obstacledistance=int(Y); //Y軸距離為機器人與障礙物實際距離
    printf("實際座標(X,Y) : (%f,%f)\n距離值 : %d\n",X,Y,obstacledistance);
}
```

(4) 安全距離副程式

當求得距離小於等於安全距離值後，進行避障。無則持續向前直到小於等於安全距離。

```
void Safedistance(int a) //安全距離副程式
{
    if(obstacledistance <= a)
    //如果機器人與障礙物實際距離小於等於a(單位為cm)，跳出該副程式
    {
    }
    else
    {
        while(obstacledistance > a)
        //如果機器人與障礙物實際距離大於a，進入步進偵測
        {
            printf("-----疑似發現障礙物-----\n");
            robot.move(50); //每次移動50mm
            if((totaldistanceB == 0)&(totaldistance >= 0))
            {
                totaldistance = totaldistance + 50;
            }
            if(totaldistanceB != 0)
            {

```


(6)避障動作副程式

避障的主要動作

```
void avoidobstacle()
{
    printf("-----進入避障動作-----\n");

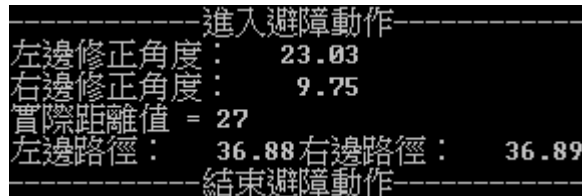
    calculateangle(l2x,l2y,r2x,r2y);           //給予左下點、右下點算出修正角度
    transformcoordinate(l2x,l2y,r2x,r2y);
    //給予左下點、右下點算出左邊點與右邊的的距離值
    LeftorRight(mlx);
    //給予中心點判斷障礙物偏左還是右

    if(leftORright==1)                         //當障礙物偏左，表示障礙物右邊較寬
    {
        turn((nowangle-(minangle+20)),5000); //先轉右邊修正角度
        robot.move(obstacledistance*10);     //移動障礙物與機器人之距離
        ArUtil::sleep(3000);

        turn((nowangle+(maxangle+20)),5000); //後轉左邊修正角度
        robot.move(obstacledistance*10);     //移動障礙物與機器人之距離
        ArUtil::sleep(3000);
    }
    if(leftORright==-1)                        //當障礙物偏右，表示障礙物左邊較寬
    {
        turn((nowangle+(maxangle+20)),5000); //先轉左邊修正角度
        robot.move(obstacledistance*10);
        ArUtil::sleep(3000);

        turn((nowangle-(minangle+20)),5000); //後轉右邊修正角度
        robot.move(obstacledistance*10);
        ArUtil::sleep(3000);
    }
    leftORright=0;
    turn(nowangle,5000);                       //回歸避障前角度
    printf("-----結束避障動作-----\n");
    takepicture();
}
```

執行畫面：



```
-----進入避障動作-----
左邊修正角度: 23.03
右邊修正角度: 9.75
實際距離值 = 27
左邊路徑: 36.88 右邊路徑: 36.89
-----結束避障動作-----
```

圖 3.21 進入避障動作顯示畫面

(7)計算左右邊修正角度

計算避障所需要之變動角度。

```
void calculateangle(int a,int b,int c,int d)
{
    maxangle=(90-(atan(double(abs(480-b))/double(abs(320-a)))*(180/M_PI)));
    printf("左邊修正角度: %8.2f\n",maxangle);
    minangle=(90-(atan(double(abs(480-d))/double(abs(c-320)))*(180/M_PI)));
    printf("右邊修正角度: %8.2f\n",minangle);
    realobstacledistance=abs(obstacledistance*cos(double(nowangle-(minangle+20))*(M_PI/180)));
    //計算較準確機器人與障礙物之距離
    printf("實際距離值= %d\n",realobstacledistance);
}
```

(8)判斷障礙物偏左還偏右

判斷障礙物偏左還偏右，機器人將會走較寬的方向進行避障。

```
void LeftorRight(int a)
{
    if(a <= 320)           //中心點小於等於半個影像的寬度，表示障礙物偏左
    {
        leftORright=1;     //以值為表示，表示障礙物偏左
    }
    if(a > 320)           //中心點大於半個影像的寬度，表示障礙物偏右
    {
        leftORright=-1;    //以-1值為表示，表示障礙物偏左
    }
}
```

(9)a 秒計時器

計時器，用於呼叫攝影機照像。

```
void timer(int a)
{
    clock_t start_time;
    start_time = clock();
    while((clock() - start_time) < a * CLOCKS_PER_SEC){}
    pictimes=1;           //完成計時器回傳
}
```

3.11 目標追尋專案介紹

3.11.1 目標追尋機器人動作主程式

```
while (Aria::getRunning()) //只要跟機器人處於連接狀態，Aria::getRunning()會持續回傳True
{
    movePos(1000, 0, 8000); //移動至座標點(0, 1000)，睡眠時間為ms
    turn(-90,5000); //讓機器人旋轉至絕對角度，大小值為-90度，睡眠時間為ms
    int n=1; //第一圈全角度目標搜尋
    turncircle(); //判斷第一圈全角度是否搜尋到目標物。是，結束程式。否，進入追加動作搜尋。
    while(z==0)
    {
        if(suma>sumb) //判斷前一搜尋之左右兩側哪邊有目標物的可能性較大，則向該邊進入額外搜尋動作
        {
            suma=0; //重製收取的兩個參數
            sumb=0;
            printf("右邊白色總面積較大,機器人往右公分搜尋\n");
            turn(-90,5000); //機器人面相右方(-90度)
            robot.move(600); //前進600mm
            ArUtil::sleep(8000);
            turn(180,5000); //轉向下方(180度)
            startmoving=1; //Reset判斷值
            angle5=0; //Reset角度參數
            angle5555=0;
            turncircleright(); //進入向右額外搜尋動作
        }
        if(sumb>suma)
        {
            suma=0;
            sumb=0;
            printf("左邊白色總面積較大,機器人往左公分搜尋\n");
            turn(90,5000); //機器人面相左方(90度)
            robot.move(600);
            ArUtil::sleep(8000);
            turn(0,5000); //轉向上方(0度)
            startmoving=1; //Reset判斷值
            angle5=0;
            angle5555=0;
            turncircleleft(); //進入向左額外搜尋動作
        }
    }
    Aria::shutdown(); //關閉ARIA所有的處理緒/執行緒並且退出程式
    return 0;
}
```

執行畫面：



```
目前角度：-70
左邊面積參數0 右邊面積參數99255
```

圖 3.22 當前角度與累計面積畫面

3.11.2 副程式

(1)全角度搜尋

```
void turncircle() //全角度搜尋之副程式
{
    while(startmoving==1)
    {
        v=0; //Reset影像處理參數
        angle5555=-90+angle5; // -90為起始之角度，每當此迴圈跑一次(即未發現目標物)，下次角度為起始角+疊加角度
        ArUtil::sleep(3000);
        takepicture(); //呼叫攝影機拍照進行影像處理
        if(z==1) //影像處理後發現目標物
    }
}
```

```

{
    startmoving=0;           //給予判斷值，動作結束後跳出迴圈
    calculatedistance(dlx,dly); //給予抵達點計算目標物與機器人之與角度差
                                //補正機器人該角度，讓機器人正對目標物
    Judgex(dlx);           //前進該距離，到達目標物前方
                                //完成動作
    v=0;                   //Reset影像處理參數
}
if(v==1)                  //影像處理過且未發現目標物
{
    turn(angle5555,3000);    //轉angle5555度，睡眠時間ms
    printf("目前角度：%d\n",angle5555);
    printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
}
n++;
if(angle5==360)          //當angle5為360度時(亦即機器人轉了一圈，回到起始角)
{
    printf("未偵測到障礙物\n");
    printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
    startmoving=0;        //給予判斷值，跳出迴圈
}
angle5=angle5+10;       //angle5疊加，設定一次加角度
}
}

```

執行畫面：

```

補正： 25.58
投影座標點<141 , 106>
以機器人為中心點之投影座標點<-179 , 374>
實際座標<X,Y> : <3.701129,52.434546>
距離值：52.565007

```

圖 3.23 發現目標物之畫面

(2)向左額外搜尋

全角度搜尋後， $0^{\circ} \sim 180^{\circ}$ 區域的面積大於 $-180^{\circ} \sim 0^{\circ}$ 區域的面積時，執行之。

```

void turncircleleft()      //向左額外搜尋之副程式
{
    while(startmoving==1)
    {
        v=0;
        angle5555=0+angle5; //0為起始之角度，每當此迴圈跑一次(即未發現目標物)，下次角度為起始角+疊加角度
        ArUtil::sleep(3000);
        takepicture();      //呼叫攝影機拍照進行影像處理
        if(z==1)           //影像處理後發現目標物
        {
            startmoving=0; //給予判斷值，動作結束後跳出迴圈
            calculatedistance(dlx,dly); //給予抵達點計算目標物與機器人之與角度差
                                        //補正機器人該角度，讓機器人正對目標物
            Judgex(dlx);     //前進該距離，到達目標物前方
                                        //完成動作
            v=0;           //Reset影像處理參數
        }
        if(v==1)          //影像處理過且未發現目標物
        {
            turn(angle5555,3000);    //轉angle5555度，睡眠時間ms
            printf("目前角度：%d\n",angle5555);
            printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
        }
        n++;
        if(angle5==180)    //當angle5為180度時(由於第一圈的結果已經確認目標物一定在機器人左

```

方，故額外搜尋只需搜尋半圈即可)

```
{
    printf("未偵測到障礙物\n");
    printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
    startmoving=0; //給予判斷值，跳出迴圈
}
angle5=angle5+10; //angle5疊加，設定一次加角度
}
```

(3)向右額外搜尋

全角度搜尋後， $-180^{\circ} \sim 0^{\circ}$ 區域的面積大於 $0^{\circ} \sim 180^{\circ}$ 區域的面積時，執行之。

```
void turncircleright() //向左額外搜尋之副程式
{
    while(startmoving==1)
    {
        v=0;
        angle5555=-180+angle5; // -180為起始之角度，每當此迴圈跑一次(即未發現目標物)，下次角度為起始角+疊加
        ArUtil::sleep(3000);
        takepicture(); //呼叫攝影機拍照進行影像處理
        if(z==1) //影像處理後發現目標物
        {
            startmoving=0; //給予判斷值，動作結束後跳出迴圈
            calculatedistance(dlx,dly); //給予抵達點計算目標物與機器人之與角度差
            //補正機器人該角度，讓機器人正對目標物
            Judgex(dlx); //前進該距離，到達目標物前方
            //完成動作
            v=0; //Reset影像處理參數
        }
        if(v==1) //影像處理過且未發現目標物
        {
            turn(angle5555,3000); //轉angle5555度，睡眠時間ms
            printf("目前角度：%d\n",angle5555);
            printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
        }
        n++;
        if(angle5==180) //當angle5為度時(由於第一圈的結果已經確認目標物一定在機器人之右方，故額外搜尋只需搜尋半圈即可)
        {
            printf("未偵測到障礙物\n");
            printf("左邊面積參數%d 右邊面積參數%d \n", sumb , suma);
            startmoving=0; //給予判斷值，跳出迴圈
        }
        angle5=angle5+10; //angle5疊加，設定一次加角度
    }
}
```

3.12 影像處理程式內容介紹

(1) 計算 $\cos \theta$

```
double angle( CvPoint* pt1, CvPoint* pt2, CvPoint* pt0 )
{
    //計算  $\theta$ 

    double dx1 = pt1->x - pt0->x;
    double dy1 = pt1->y - pt0->y;
    double dx2 = pt2->x - pt0->x;
    double dy2 = pt2->y - pt0->y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
}
```

(2) 處理影像是否為矩形(目標物)及座標點和判斷面積

```
CvSeq* findSquares4( IplImage* img, CvMemStorage* storage )
{
    CvSeq* contours; //CvSeq類似C++的Array
    int i, c, l, N = 11;
    CvSize sz = cvSize( img->width & -2, img->height & -2 ); //初始化CvSize資料結構,分別填入int型別的寬度跟高度數據
    IplImage* timg = cvCloneImage( img ); //二值化後的圖放到cpy
    IplImage* gray = cvCreateImage( sz, 8, 1 ); //cvCreateImage(CvSize圖形大小資料結構,IplImage格式參數,通道數)
    IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
    IplImage* tgray;
    CvSeq* result;
    double s, t;

    CvSeq* squares = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvPoint), storage ); //建立一個空序列存儲每個四邊形的四個頂點
    //設定timg的ROI為最大值 ( )
    cvSetImageROI( timg, cvRect( 0, 0, sz.width, sz.height ) );
    tgray = cvCreateImage( sz, 8, 1 ); //濾除雜訊
    for( c = 0; c < 3; c++ ) //尋找每個通道的四邊形
    { //提取第c個通道
        cvSetImageCOI( timg, c+1 ); //嘗試每個閾值等級
        cvCopy( timg, tgray, 0 );
        for( l = 0; l < N; l++ ) //Canny代替零閾值, Canny通過梯度變化程度大來尋找四邊形
        {
            if( l == 0 )
            {
                cvCanny( tgray, gray, 60, 180, 3 ); //canny邊緣檢測
                cvDilate( gray, gray, 0, 1 ); // 檢測完後膨脹
            }
            else
            {
                cvThreshold( tgray, gray, 50, 255, CV_THRESH_BINARY ); // 找到邊緣後儲存
            }
            cvFindContours( gray, storage, &contours, sizeof(CvContour),
                CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
            while( contours ) //測試每一個輪廓
            { //用指定精度逼近多邊形曲線
                result = cvApproxPoly( contours, sizeof(CvContour), storage,
                    CV_POLY_APPROX_DP, cvContourPerimeter(contours)*0.02, 0 );
                if( result->total == 4 && //計算整個輪廓或部分輪廓的面積
                    fabs(cvContourArea(result, CV_WHOLE_SEQ)) > 1000 &&
                    cvCheckContourConvexity(result) )
                {
                    s = 0;
                    for( i = 0; i < 5; i++ )
                    { //尋找介於兩個接縫邊緣的角度(maximum of cosine)
                        if( i >= 2 )
                        {
                            t = fabs(angle(
```



```

        (CvPoint*)cvGetSeqElem( result, i ),
        (CvPoint*)cvGetSeqElem( result, i-2 ),
        (CvPoint*)cvGetSeqElem( result, i-1 ));
        s = s > t ? s : t;
    }
}
if( s < 0.3 )
    for( i = 0; i < 4; i++ )
        cvSeqPush( squares,
            (CvPoint*)cvGetSeqElem( result, i ));
}

//判斷左側和右側的面積
if(angle5555<0 || angle5555==-180 || angle5555 ==180 ||(angle5555>180 && angle5555 < 270))
{
    area=fabs(cvContourArea(result,CV_WHOLE_SEQ));
    suma=suma+area;
}
if((angle5555>0 && angle5555<180) || angle5555==0)
{
    areb=fabs(cvContourArea(result,CV_WHOLE_SEQ));
    sumb=sumb+areb;
}
contours = contours->h_next; // 執行下一個輪廓
}
}
}
printf("%d\n",z);
return squares;
}

```

(3)讀取四點座標並標記及判斷是否為障礙物

```

void drawSquares( IplImage* img, CvSeq* squares ) //把偵測到的矩形畫出來
{
    CvSeqReader reader;
    IplImage* cpy = cvCloneImage( img );
    int i;
    cvStartReadSeq( squares, &reader, 0 ); // 初始化序列
    for( i = 0; i < squares->total; i += 4 ) // 讀取序列中的元素(矩形的所有頂點)
    {
        CvPoint* rect = pt;
        int count = 4;
        CV_READ_SEQ_ELEM(pt[0],reader); // 讀四頂點
        CV_READ_SEQ_ELEM(pt[1],reader);
        CV_READ_SEQ_ELEM(pt[2],reader);
        CV_READ_SEQ_ELEM(pt[3],reader);

        // 畫矩形
        cvPolyLine( cpy, &rect, &count, 1, 1, CV_RGB(255,0,0), 2, CV_AA, 0 );
    }

    //判斷是否有障礙物
    for(int i=0;i<6;i++)
    {
        if((pt[i].x&pt[i].y)>0)
        {
            k++;
        }
        if(k==5)
        {
            z=1;
        }
    }
    cvShowImage(" wndname", cpy );
    cvSaveImage("Square.jpg",cpy);
}

```

//顯示四點座標

```
printf("(%d , %d)",pt[0].x,pt[0].y);
printf("\n");
l2x=pt[0].x;
l2y=pt[0].y;
printf("(%d , %d)",pt[1].x,pt[1].y);
printf("\n");
l1x=pt[1].x;
l1y=pt[1].y;
printf("(%d , %d)",pt[2].x,pt[2].y);
printf("\n");
r1x=pt[3].x;
r1y=pt[3].y;
printf("(%d , %d)",pt[3].x,pt[3].y);
printf("\n");
r2x=pt[4].x;
r2y=pt[4].y;
```

//計算中心點X,Y

```
pt[5].x=(pt[0].x+pt[1].x+pt[2].x+pt[3].x)/4;
pt[5].y=(pt[0].y+pt[1].y+pt[2].y+pt[3].y)/4;
mlx=pt[5].x;
mly=pt[5].y;
if(mlx!=0&mly!=0)
{
    z=1;
    printf("(中心點%d , %d)",mlx,mly);
    printf("\n");
}
int g=0;
for(int r=0;r<7;r++)
{
    if(pt[r].y>g)
    {
        g=pt[r].y;
    }
}
if(z==1)
{
    pt[6].x=pt[5].x;
    pt[6].y=g+10;
    printf("(抵達點:%d , %d)\n",pt[6].x,pt[6].y);
}else
{
    pt[6].y=0;
}
dlx=pt[6].x;
dly=pt[6].y;
corner();
}
```

(4)呼叫攝影機中擷取影像且展示

```
void takepicture()
{
    capture = cvCaptureFromCAM(0); //擷取圖片
    Image1 =cvQueryFrame(capture);
    Image2=cvCloneImage( Image1); //複製抓到的影像
    cvNamedWindow("Webcam",0); //創建新視窗
    cvShowImage("Webcam",Image1); //展示圖片
    wabcam();
}
```

(5) 影像處理流程

```
void wabcam() //影像處理
{
    gray(); //灰階
    two(); //二值
    filter(); //濾波
    sobel(); //邊緣檢測
    compare2(); //比較&合併
    ED(); //侵蝕與擴張
    compare(); //比較&合併
    mark(); //濾掉不連續點(雜質)
    ED2(); //侵蝕與擴張
    cvWaitKey(100); //延遲ms
    v=1;
}
```

(6) 灰階處理

將影像的 RGB 值分別乘上 0.229 0.587 0.114 並儲存

```
void gray() //灰階
{
    Height=cvGetDimSize(Image1,0); //自動抓影像高度參數為"0"
    Width=cvGetDimSize(Image1,1); //自動抓影像寬度參數為"1"

    for(int i=0;i<Image1->height;i++) //讀取影像RGB值
    {
        for(int j=0;j<Image1->widthStep;j=j+3)
        {
            Blue[i][j/3]=Image1->imageData[i*Image1->widthStep+j];
            Green[i][j/3]=Image1->imageData[i*Image1->widthStep+j+1];
            Red[i][j/3]=Image1->imageData[i*Image1->widthStep+j+2];
        }
    }
    for(int i=0;i<Image1->height;i++) //灰階處理
    {
        for(int j=0;j<Image1->width;j++)
        {
            Gray[i][j]=(uchar)(0.299*Red[i][j] + 0.587*Green[i][j] + 0.114*Blue[i][j]);
            Red[i][j]=Gray[i][j];
            Green[i][j]=Gray[i][j];
            Blue[i][j]=Gray[i][j];
        }
    }
    for(int i=0;i<Image1->height;i++) //儲存新的RGB值(灰階後)
    {
        for(int j=0;j<Image1->widthStep;j=j+3)
        {
            Image1->imageData[i*Image1->widthStep+j]=Blue[i][j/3];
            Image1->imageData[i*Image1->widthStep+j+1]=Green[i][j/3];
            Image1->imageData[i*Image1->widthStep+j+2]=Red[i][j/3];
        }
    }
    Image2=cvCloneImage(Image1);
    cvSaveImage("gray.jpg",Image1); //儲存影像
}
```

(7) 二值化

將子像素作判斷 75 以上設為 0，以下則設為 255。

```
void two() //二值化
{ //將灰階值在75以上設為0,以下則設為255
    cvThreshold(Image2,Image2,75,255,CV_THRESH_BINARY_INV);
    cvSaveImage("two-a.jpg",Image2);
}
```

(8) 均值濾波

將每點以九宮格的形式相加並除以九且回存。

```
void filter() //濾波器
{
    Image1 = cvLoadImage("gray.jpg");
    Height=cvGetDimSize(Image1,0); //自動抓影像高度參數為"0"
    Width=cvGetDimSize(Image1,1); //自動抓影像寬度參數為"1"
    for(int i=0;i<Image1->height;i++)
    {
        for(int j=0;j<Image1->widthStep;j=j+3)
        {
            B[i][((int)(j/3))]=Image1->imageData[i*Image1->widthStep+j];
            G[i][((int)(j/3))]=Image1->imageData[i*Image1->widthStep+j+1];
            R[i][((int)(j/3))]=Image1->imageData[i*Image1->widthStep+j+2];
        }
    }
    for (int i = 0 ; i <Height-1 ; i++) //影像處理 (均值濾波)
    {
        for (int j = 0; j <Width-1 ; j++)
        {
            R[i][j] = ( R[i-1][j-1] + R[i-1][j] + R[i-1][j+1] +
                R[i][j-1] + R[i][j] + R[i][j+1] +
                R[i+1][j-1] + R[i+1][j] + R[i+1][j+1] ) / 9;
            G[i][j] = ( G[i-1][j-1] + G[i-1][j] + G[i-1][j+1] +
                G[i][j-1] + G[i][j] + G[i][j+1] +
                G[i+1][j-1] + G[i+1][j] + G[i+1][j+1] ) / 9;
            B[i][j] = ( B[i-1][j-1] + B[i-1][j] + B[i-1][j+1] +
                B[i][j-1] + B[i][j] + B[i][j+1] +
                B[i+1][j-1] + B[i+1][j] + B[i+1][j+1] ) / 9;
        }
    }
    //儲存新的RGB值(濾撥後)
    for(int i=0;i<Image1->height;i++)
    {
        for(int j=0;j<Image1->widthStep;j=j+3)
        {
            Image1->imageData[i*Image1->widthStep+j]=B[i][((int)(j/3))];
            Image1->imageData[i*Image1->widthStep+j+1]=G[i][((int)(j/3))];
            Image1->imageData[i*Image1->widthStep+j+2]=R[i][((int)(j/3))];
        }
    }
    cvSaveImage("filter.jpg",Image1); //儲存處理結果至新的圖檔中
}
```

(9) 邊緣檢測

```
void sobel() //sobel邊緣檢測
{
    IplImage *image;
    IplImage *src;
    IplImage *dst;
    IplImage *dstb;
    bool createMemory = false;
    while(1)
    {
        image = cvLoadImage("filter.jpg");
        if(!createMemory)
        {
            src = cvCreateImage( cvGetSize(image), image->depth, 1); //影像來源只接受-bit
            dst = cvCreateImage( cvGetSize(image), IPL_DEPTH_16S, 1); //影像輸出只接受-bit，因為會有負值
            dstb = cvCreateImage( cvGetSize(image), IPL_DEPTH_16S, 1);
            dst_8U = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1); //影像最後須轉乘-bit顯示，影像只能顯示~255的值
        }
    }
}
```

```

    dstb_8U = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1);
    createMemory = !createMemory; //記憶體宣告完成
}
cvCvtColor( image, src, CV_BGR2GRAY);
cvSobel(src, dst, 1,0, 3); //針對X方向的Sobel計算
cvSobel(src, dstb, 0,1, 3); //cvSobel( 影像來源, 影像輸出, X方向的差分, Y方向
                             的差分, kernel )

cvConvertScaleAbs( dst, dst_8U, 1, 0);
cvConvertScaleAbs( dstb, dstb_8U, 1, 0);

//影像最後需靠cvConvertScale或cvConvertScaleAbs
//轉換影像格式才可輸出

cvSaveImage("Sobel-X.jpg",dstb_8U);
cvSaveImage("Sobel-Y.jpg",dst_8U);
break;
}
}

```

(10) 侵蝕與擴張

先侵蝕再在膨脹則斷開，先膨脹再侵蝕則閉合，能有效去除大量的雜質。

```

void ED() //侵蝕與擴張
{
    IplImage *src, *dst;
    IplConvKernel * pKernel = NULL;
    src = cvLoadImage( "two-b.jpg", 0);
    dst = cvCreateImage(cvSize(src->width, src->height), src->depth, 1);
    DilateImg = cvCreateImage(cvSize(src->width, src->height), src->depth, 1);
    ErodeImg = cvCreateImage(cvSize(src->width, src->height), src->depth, 1);
    int pos = 3;
    pKernel =NULL;

    cvErode(src, ErodeImg, pKernel, 1); //斷開(Opening)先侵蝕再在膨脹
    cvDilate(ErodeImg, DilateImg, pKernel, 1); //cvErode(來源影像,輸出影像,侵蝕結構,侵蝕次數);
    cvSaveImage("Dilate.jpg", DilateImg); //cvDilate(來源影像,輸出影像,膨脹結構,膨脹次數);
    cvSaveImage("Erode.jpg", ErodeImg);

    //閉合(Closing)先膨脹再在侵蝕
    cvDilate(DilateImg, DilateImg, pKernel, 1);
    cvErode(DilateImg, ErodeImg, pKernel, 1);
    cvSaveImage("Dilate-a.jpg", DilateImg);
    cvSaveImage("Erode-a.jpg", ErodeImg);
}

```

(11) 比較圖片並合併

將兩張圖片做 AND 合併。

```

void compare() //比較圖片並合併
{
    IplImage *src,*dst,*dst_temp;
    src=cvLoadImage("two-a.jpg",1);
    dst=cvLoadImage("Dilate.jpg",1);
    dst_temp=cvCreateImage(cvGetSize(src),src->depth,src->nChannels);
    cvResize(dst,dst_temp);
    cvAnd(src,dst_temp,src); //兩個數據中的元素做And合併
    cvSaveImage("comb.jpg",src);
    cvThreshold(src,src,254,255,CV_THRESH_BINARY);
    cvSaveImage("comb.jpg",src);
}

```

(12) 濾掉不連續點(雜質)

將每點 RGB 值左右各 35 點做判斷是否連續，否為 0，是為 255。

```
void mark() //濾掉不連續點(雜質)
{
    int a,b; //height 0~479
    int y=0; //width 0~639
    Image0= cvLoadImage("comb.jpg");
    for(int i=0;i<Image0->height;i++)
    {
        for(int j=0;j<Image0->widthStep;j=j+3)
        {
            B[i][ (int)(j/3)]=Image0->imageData[i*Image0->widthStep+j];
            G[i][ (int)(j/3)]=Image0->imageData[i*Image0->widthStep+j+1];
            R[i][ (int)(j/3)]=Image0->imageData[i*Image0->widthStep+j+2];
        }
    }
    for (int i = 0 ; i <Height-1 ; i++) //濾掉不連續點(雜質)
    {
        for (int j = 0; j <Width-1 ; j++)
        {
            if(R[i][j]==255||B[i][j]==255||G[i][j]==255)
            {
                for(int k=1;k<35;k++)
                {
                    if(R[i][j+k]==255||B[i][j+k]==255||G[i][j+k]==255||R[i][j-k]==255||B[i][j-k]==255||G[i][j-k]=
                    =255)
                    {
                        R[i][j]=255;
                        B[i][j]=255;
                        G[i][j]=255;
                    }
                    else
                    {
                        R[i][j]=0;
                        B[i][j]=0;
                        G[i][j]=0;
                    }
                }
            }
            else
            {
                R[i][j]=0;
                B[i][j]=0;
                G[i][j]=0;
            }
        }
    }
    for(int i=0;i<Image0->height;i++)
    {
        for(int j=0;j<Image0->widthStep;j=j+3)
        {
            Image0->imageData[i*Image0->widthStep+j]=B[i][ (int)(j/3)];
            Image0->imageData[i*Image0->widthStep+j+1]=G[i][ (int)(j/3)];
            Image0->imageData[i*Image0->widthStep+j+2]=R[i][ (int)(j/3)];
        }
    }
    cvSaveImage("abb.jpg", Image0);
}
```

(13)標記座標點

在圖像中標記座標點。

```
void corner() //標記座標點
{
    IplImage *src, *dst;
    int k=0;
    cvNamedWindow("Result2", 1);
    src = cvLoadImage("Dilate-2.jpg");
    dst = cvCloneImage(src);
        for(int i = 0 ; i < 10 ; ++i)
            {
                cvLine( dst, cvPoint(pt[i].x, pt[i].y),
                    cvPoint(pt[i].x, pt[i].y), CV_RGB(255,0,0), 5);
            }
    cvShowImage("Result2", dst);
    cvWaitKey(50);
        for(int i = 0 ; i < 10 ; ++i) //清除序列
            {
                pt[i].x=0;
                pt[i].y=0;
            }
}
```

第 4 章 模擬與實現

4.1 無線導航機器人之模擬

我們先畫出電機系二樓左側地圖。圖 4.1 為國立虎尾科技大學電機系二樓左側的環境示意圖，也是本次專題的實驗環境。

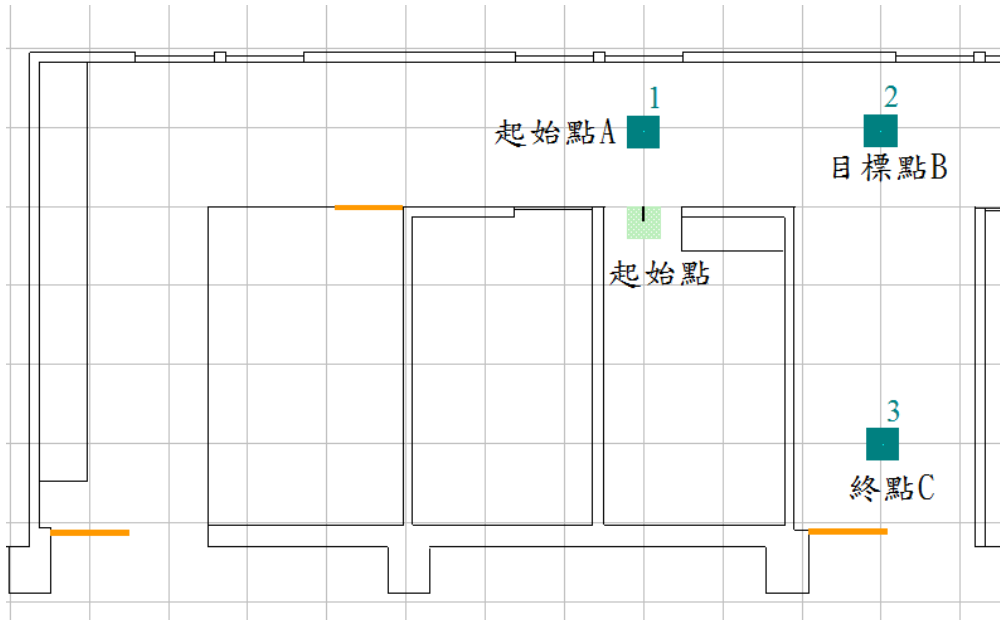


圖 4.1 國立虎尾科技大學電機系二樓左側

我們畫好地圖之後，使用軟體模擬，我們先讓無線導航機器人開啟，讓它開始作偵測動作，先到起始點 A，到達起點後執行走到目標點 B 到達 B 點後前往終點 C，然後停止。下圖為軟體模擬圖：

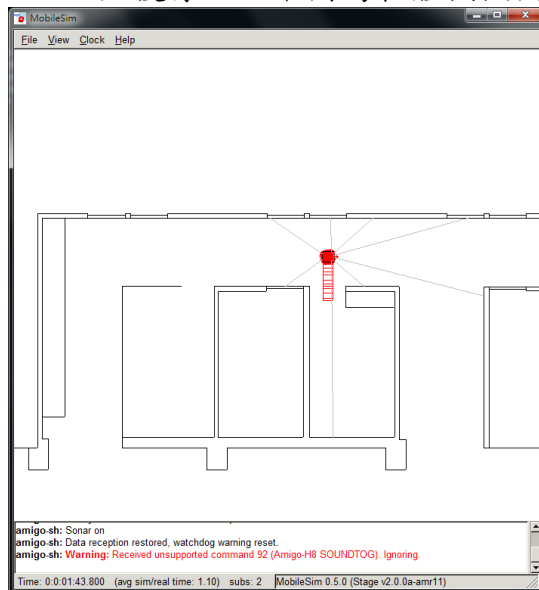


圖 4.2 抵達起點 A

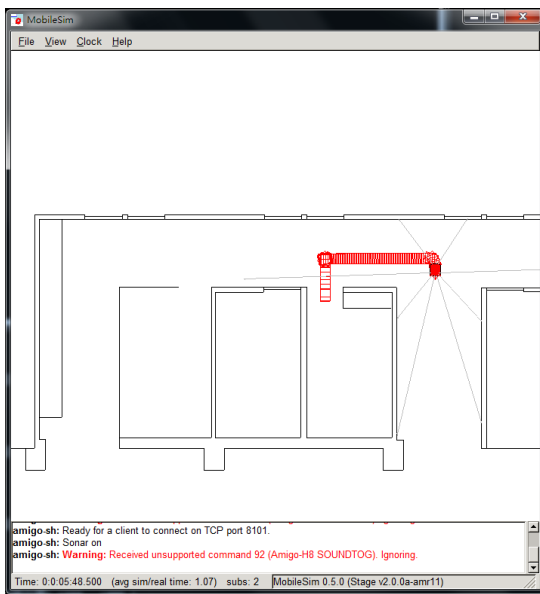


圖 4.3 抵達目標點 B 前往終點 C

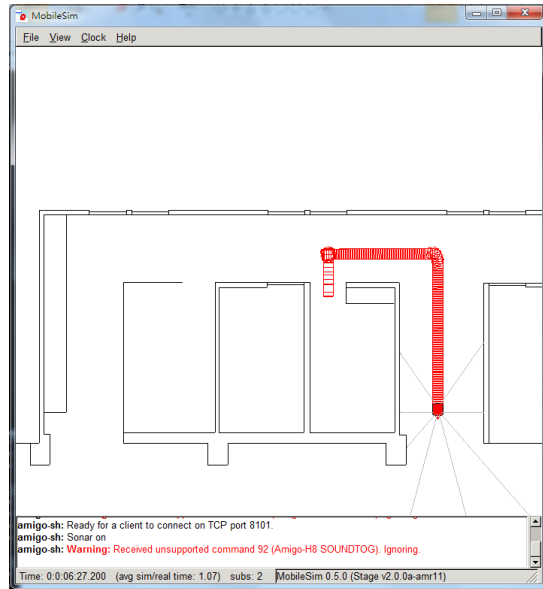


圖 4.4 抵達終點 C 並停止

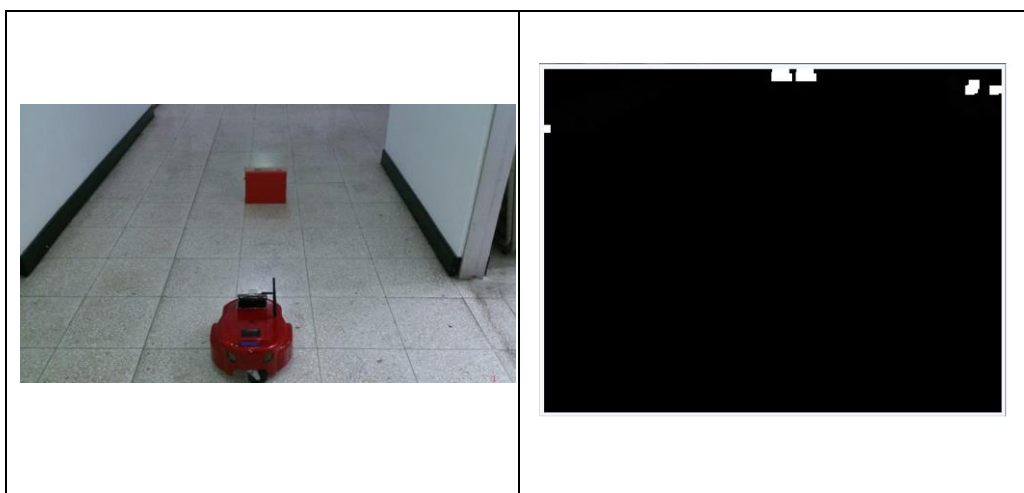
4.2 整體實驗結果

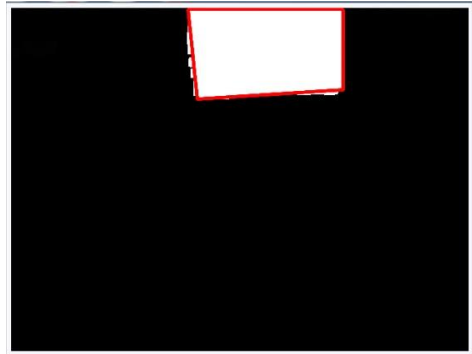
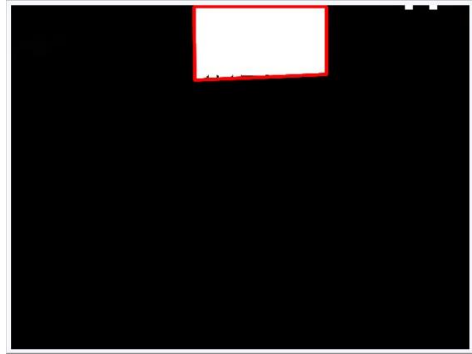
我們在學校走廊作實際測試，以下分成兩部分呈現。

4.2.1 閃避障礙物與前往路徑點結果

本專題所設計的無線導航機器人，應用在室內環境中利用影像了解周圍環境。但是在室內中行走可能會遇到障礙物干擾，所以閃避障礙物是相當重要的。我們在以下顯示了閃避障礙物與影像處理過後影像。

圖 4.5 室內閃避障礙物程序圖，一開始為正常行走，當遇到障礙物時他會判斷障礙物偏左或右選擇較大空間的走道通行，以下範例可以知道他往左邊行走因障礙物偏右，他會先往左方旋轉補正角度，等與障礙物平行後再往右方旋轉補正角度，最後回歸路徑，完成閃避障礙物再繼續行走。





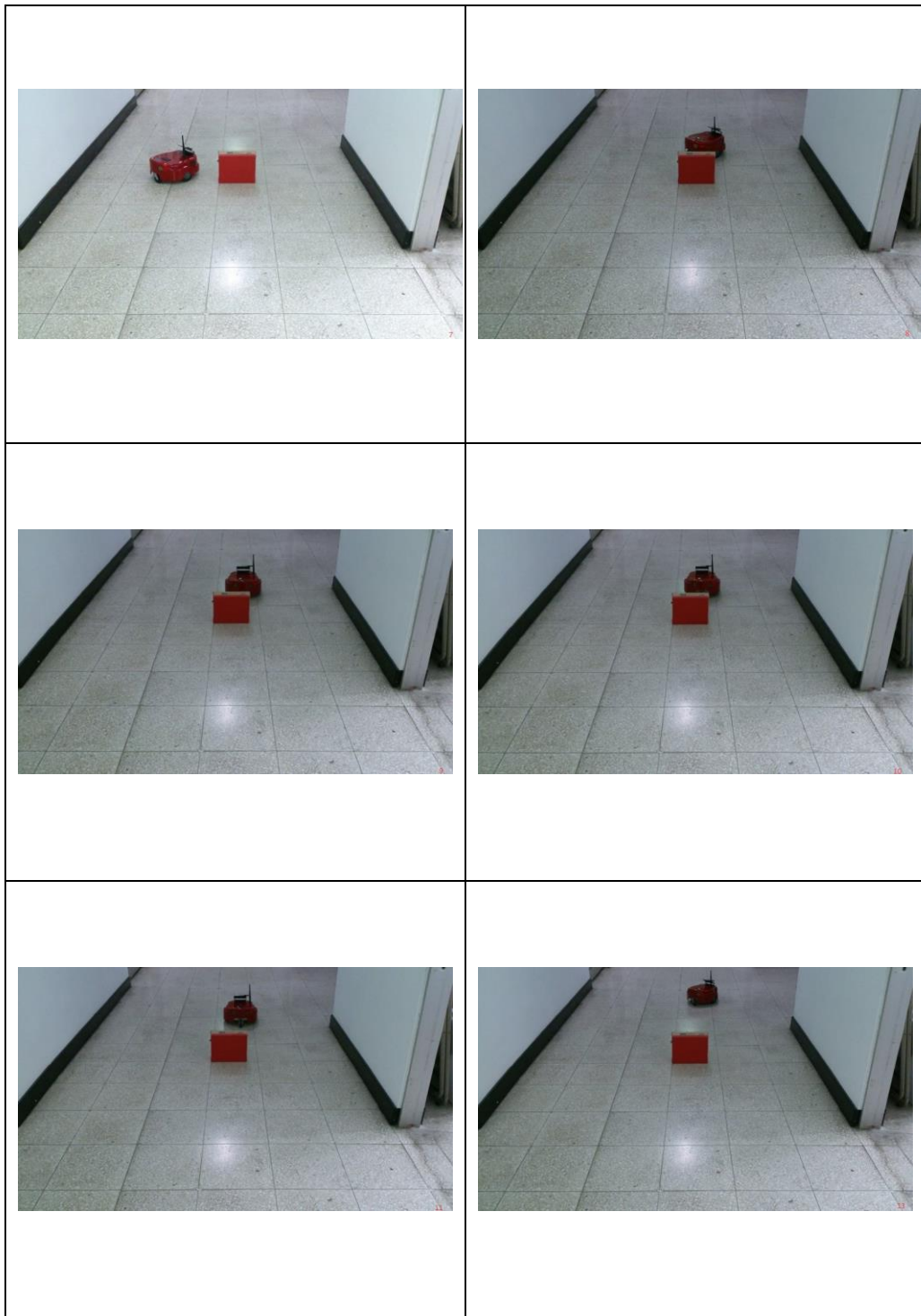


圖 4.5 室內閃避路徑上障礙物程序圖與影像處理圖

圖 4.6 接續上一部之動作，當機器人抵達目標點 B 後將前往終點 C 之程序圖。因 B 點至 C 點無任何障礙物，機器人將以原規劃路徑前往終點 C 至完成巡邏工作。

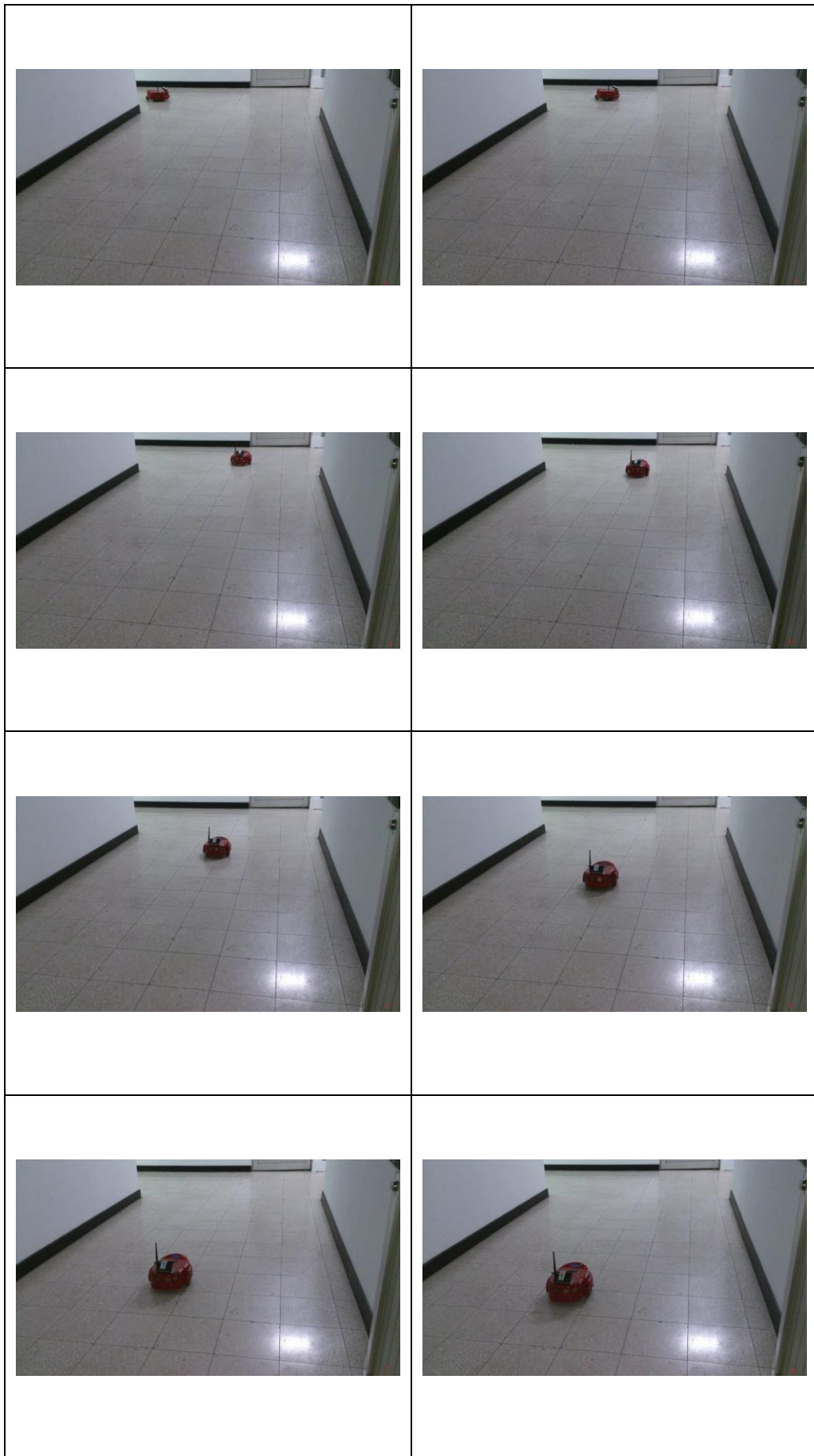
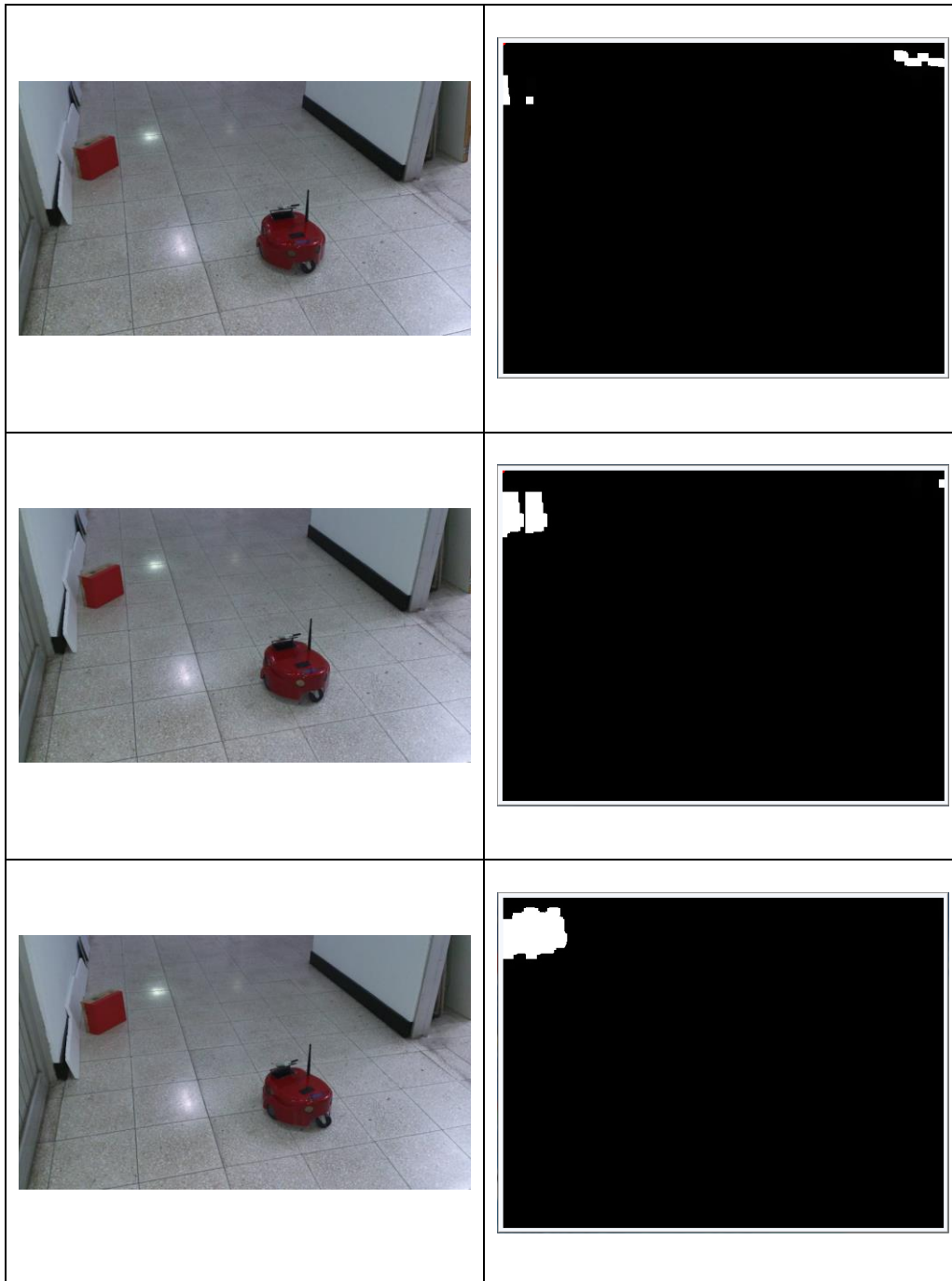


圖 4.6 以原規劃路徑前往終點 C 程序圖

4.2.2 目標追尋結果

本專題所使用之攝影機是固定於機器人之上且向下斜拍於地面，經過測試，目標物與機器人之距離約 1.5 公尺內可完整判別，而攝影機最遠可視距離約 2.3 公尺。換句話說，如目標物位於相對機器人 1.5 至 2.3 公尺可能判別不到，必須利用額外搜尋功能完成任務；但目標物相對於機器人超出 2.3 公尺，機器人無跡可尋，無法完成任務。

圖 4.7 為目標物位於 1.5 公尺內，機器人動作與影像處理圖。機器人位於指定位置後，每向左旋轉 10 度拍攝圖片進行影像處理以找出目標物。若處理後無發現目標物，繼續向左 10 度搜尋…。



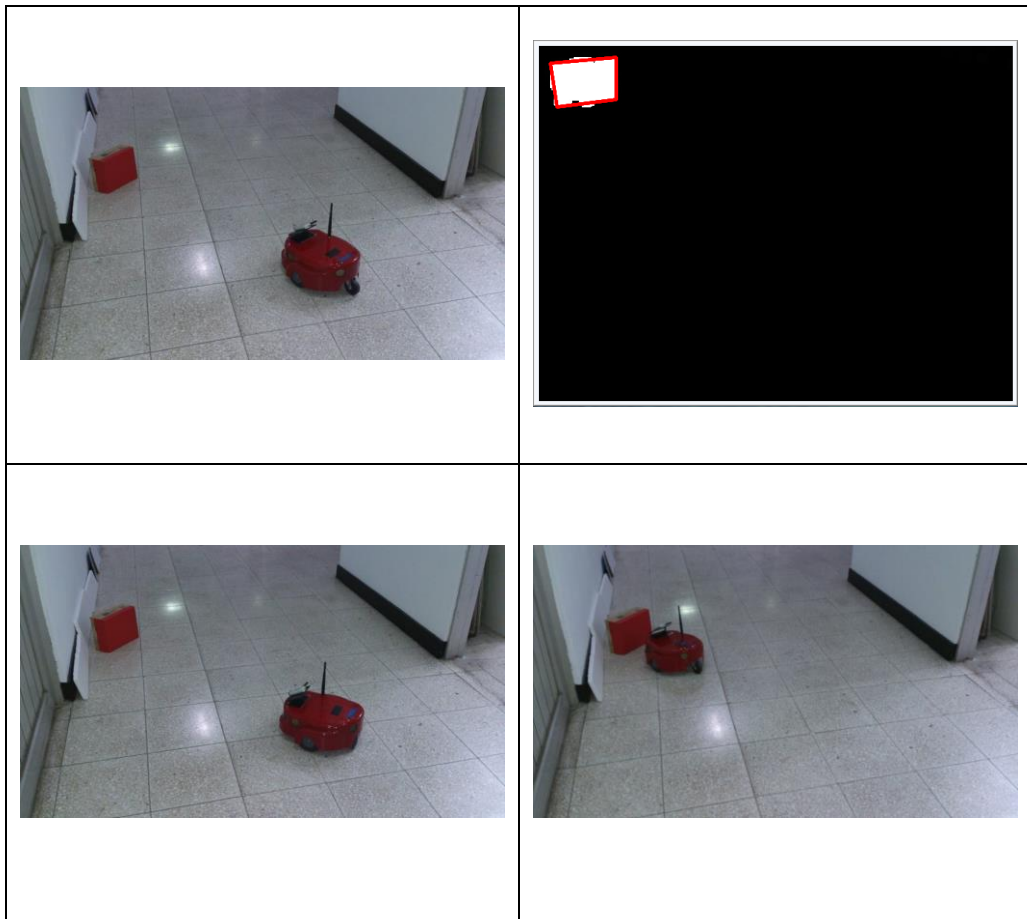


圖 4.7 目標物於 1.5 公尺內目標追尋程序與影像處理圖

圖 4.8 為目標物介於 1.5 公尺至 2.3 公尺內，機器人額外搜尋之動作程序圖與影像處理圖。由於全角度的尋找結果已知道目標物可能位於前方，故完成全角度搜尋後向前移動 60 公分後只搜尋前方，以節省動作時間。



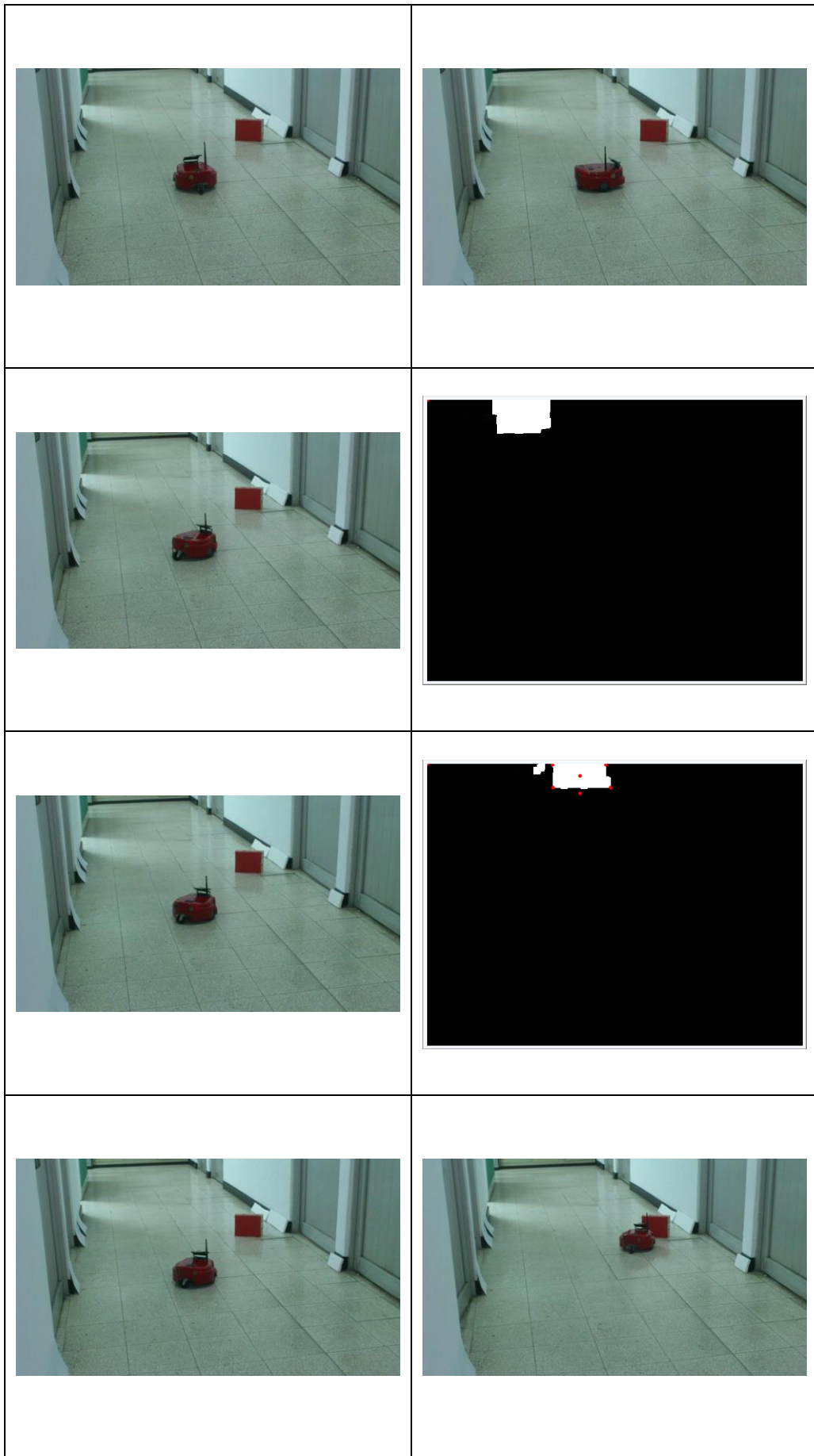


圖 4.8 目標物於 1.5 公尺至 2.3 公尺內目標追尋程序與影像處理圖

第 5 章 結論與未來展望

5.1 結論

本專題利用 Microsoft Visual C++ 作為程式編譯器，引用 ARIA 函式庫和 OpenCV 函式庫作為無線導航機器人和影像處理控制指令下達，讓無線導航機器人在室內中執行巡邏且具有避障功能以及搜尋目標的目標追尋功能。在影像處理方面，我們使用二值化、濾波、邊緣檢測、擴張、侵蝕等方法，消除室內燈光或其他雜訊干擾，再使用四角形的幾何特性來辨別障礙物以及目標物，標示出該物體各四角之影像點，運用轉換法求出距離值，角度…等等。作為避障動作或目標追尋動作之依據。

5.2 未來展望

本專題所提出之方法以確實完成，但目前所發展的方法在功能上仍未盡完備，因此在日後的研究上來加以改善，使功能更加完善。

在本次主控端是使用個人電腦做影像處理與控制命令下達，但個人電腦也同時處理其它作業系統，拖累執行速度及記憶體空間，未來可以改用專門為影像處理開發的 DSP 或 FPGA 晶片來加快速度及節省體積。

無線導航機器人行走比較長的距離後，容易因本身的機器誤差影響，造成他行走的偏差，未來可以在這部分來做改善。避障方面，只單純以影像為依據進行判斷，也許能再加入聲納增加其精確性。或是以更精密的避障演算來完成較高難度的避障動作，也是改善之一。

可以在未來將無線導航機器人應用在居家安全方面，當家中或其他應用場所發生狀況時能傳簡訊或寄電子郵件告知使用者。在影像處理方面能再增加其他圖形辨識的能力，甚至可以再增加其他影像處理技巧，如臉部偵測等等，使功能更加完善。

參考文獻與書籍

1. 陳明哲，“利用自動車作基於室內安全巡邏”，國立交通大學資訊科學研究所，碩士論文，2004年。
2. 葉英傑，“居家保全機器人之即時影像處理技術”，國立成功大學電機工程研究所，碩士論文，2003年。
3. 賴一翔，“具搜尋與避障之自動跟隨之機器人”，國立中央大學電機工程研究所，碩士論文，2009年。
4. Gonzalez Woods，“Digital Image Processing”。
5. Alasdair McAndrew，“Introduction to Digital Image Processing with MATLAB”。
6. 鍾國亮，“數位影像處理與電腦視覺”。
7. Gary Bradski & Adrian Kaebler，“Learning OpenCV”。
8. 劉瑞禎&于仕琪，“OpenCV---基礎篇”。