國立交通大學

機械工程學系

博 士 論 文

針對動態路徑規劃之 D++演算法研究及其應用

The Research and Application of D++ Algorithm for Dynamic Path-Planning

研 究 生：陳品均
指導教授：鄭璧瑩　博士

中 華 民 國　102 年 7 月

針對動態路徑規劃之 D++演算法研究及其應用

The Research and Application of D++ Algorithm for Dynamic Path-Planning

研 究 生： 陳品均　　Student： Pin-Jyun Chen

指導教授： 鄭璧瑩 博士　　Advisor： Dr. Pi-Ying Cheng

國 立 交 通 大 學

機 械 工 程 學 系

博 士 論 文

A Dissertation

Submitted to Department of Mechanical Engineering

College of Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

PhD

in

Mechanical Engineering

June 2013

Hsinchu, Taiwan, Republic of China

針對動態路徑規劃之 D++演算法研究及其應用

研究生：陳品均　　指導教授：鄭璧瑩博士

國立交通大學機械工程學系

摘要

移動機器人的導航科技是機器人科學中一項重要的技術。在過去數十年間，移動機器人的導航技術吸引了許多學界與產業界的注意，並發展出許多的技術。在已知或未知的環境所進行的主要導航科技，包括了探索、環境建置、定位、運動控制、路徑規劃。路徑規劃技術為引導機器人從一個起點走到它的目標點。現今的主要路徑規劃大致分為全域路徑規劃法，與區域路徑規劃法。全域路徑規劃法的優點為可找出最短路徑，然而計算量龐大且耗時久，故大多用於靜態環境。區域路徑規劃法的優點為可快速更新環境資訊並找出可行路徑，故適用於動態環境。然而區域路徑規劃法的缺點為不能保證找到最短路徑，尤其是處理類似迷宮的環境，非常容易卡在某特定區域(區域解)而無法找到終點。

在本研究中，我們改良舊有的 Dijkstra 演算法，並發展成一種新的演算法：D++演算法；並且應用它在移動機器人的導航科技中。我們結合了 Dijkstra 演算法與環境感測方法，讓原本屬於全域搜尋的 Dijkstra 演算法轉變區域搜尋的 D++演算法，因此可讓機器人能即時決定下一步該怎麼移動。雖然 D++演算法大部分時間為在有限的區域進行區域搜索，但它也可以在需要的時候擴大搜索範圍以避開區域解的狀況。因此 D++演算法綜合了全域搜尋與區域搜尋的優點，不斷可快速反應處理動態環境問題，即使面對迷宮的環境，也可輕易解決區域解問題而成功找到終點。

而在本論文後面的章節，我們也應用 D++演算法在實際的移動機器人上，並使其航行在數個測試環境中。由實驗結果我們驗證了 D++演算法可有效地實現在真實的移動機器人上。因此我們可以期望未來可應用D++演算法於野外型探勘機器人，以應付複雜、未知、動態、廣大的真實環境。

關鍵字：機器人；路徑規劃；Dijkstra 演算法

The Research and Application of D++ Algorithm for Dynamic Path-Planning

Student：Cheng, Kuei-Jen Advisors：Dr. Cheng, Pi-Ying

Department of Mechanical Engineering
National Chiao Tung University

ABSTRACT

The navigation of mobile robots is a vital aspect of technology in robotics. During the past few decades, the navigation of mobile robots has attracted notable attention, and considerable research was developed. The main technology of navigation in unknown or uncertain environments includes exploration, mapping, localization, motion control, and path-planning. Path planning is one of the main technologies to direct a robot from a starting point to its destination. The main methods of path planning at present can be classified by global search type and local search type. The advantage of global search path-planning is that it can find a shortest path. However, it need plenty of computation and takes a long time. Therefore it often only use static environment. The advantage of local search path-planning is that it can find probable moving direction very fast. Therefore, it is very suitable for dynamic environment. However, the shortcoming of local search path-planning is that it can not ensure to find a shortest path, especially for some environments of maze. It is very easily to get stuck at some area which is a local solution so that it can not a path to destination eventually.

In this paper, we applied the D++ algorithm, which is a novel and improved path-planning algorithm, to the navigation of mobile robots. The D++ algorithm combines Dijkstra's algorithm with the idea of a sensor-based

method, such that Dijkstra's algorithm is adapted to local search, and the robot can determine its next move in real-time. Although the D++ algorithm frequently runs local search with limited ranges, it can compute optimum paths by expanding the size of the searching range to avoid local minima. Therefore D++ algorithm combines the advantages of global seach and local search. It not only can deal with dynamic environment rapidly, but also avoid to the problem of local solution for the environment of maze.

In the later chapters, we applied the D++ algorithm to a real mobile robot in a number of environments. According to the results of experiments we verified that the D++ algorithm is very practicability for real mobile robot. Therefore we can expect that use of the D++ algorithm enables robots to navigate efficiently in complex, unknown, dynamic and large environments.

Keywords : Robot, Path planning , Dijkstra algorithm

誌謝

在博士班求學期間，首先感謝指導教授鄭璧瑩博士這些年來的指導與教誨，使我能如期順利地完成博士論文，並學習到許多寶貴的知識與經驗。另外感謝口試委員徐瑞坤教授、吳隆庸教授、歐耿良院長、陳昭亮教授、曾釋鋒博士的許多重要意見，使得本論文的內容更加地充實、完備。

接著要感謝在求學期間，在工作上非常照顧我的長官同仁們，雷晟科技的唐沛澤總經理、程建智經理，金屬中心的黃昆明處長、吳春甫副處長、魏江銘組長，與其他曾一起共事的夥伴們。讓我可以在工作閒暇之餘，順利完成我的博士學位。

最後，論文能順利完成還要感謝我的父親陳正宏、母親徐翠蓮、姐姐陳萱蓉、與女友官愛蓮，在這段時間給予的支持與鼓勵，讓我能夠無後顧之憂的完成博士學業。以上我所感謝的人，對於你們的幫助，紙短情長，無法表達我的感激之情於萬一。僅將本論文的成果與榮耀與你們分享。

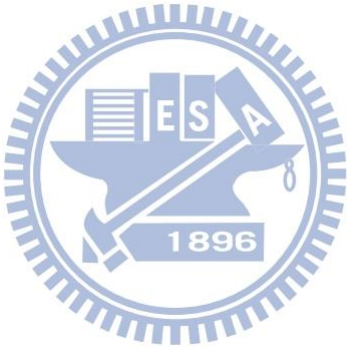
目錄

表目錄

頁次

圖目錄

頁次



一、緒論

1.1 研究背景

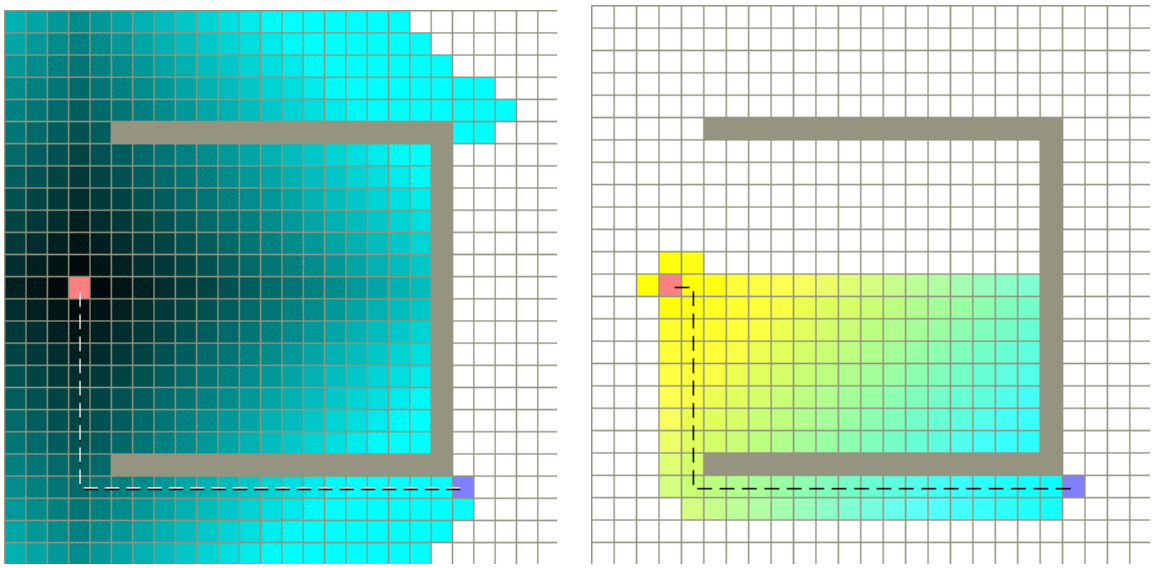
在過去的二十年中，由於人工智慧理論[1]與電腦硬體科技的快速進步，以往僅存在於電影小說的機器人科技，受到了越來越多研究人員的注意與研究，並且廣泛應用於工業製造、太空探索與軍事領域上。例如移動機器人可以替代人們探索未知與危險的環境、或是變成自動駕駛的汽車...等等。其中路徑規劃技術為移動機器人科技中，一項重要且關鍵的技術。另外除了工業、航太與軍事領域外，路徑規劃技術已成功地應用到一些日常生活產品，例如電玩遊戲、導航軟體、吸塵機器人等。

根據所需處理環境資訊的差異，路徑規劃主要可分為兩種類型：(1)全域路徑規劃：是指所有的環境資訊是已先行透過偵查或感測獲得，並且之後也不會再變動的，一般稱之為「靜態環境」。因此，機器人可輕易地透過全域路徑規劃方法，從起點找到一條最短或最佳路徑到終點。(2)區域路徑規劃：是指環境資訊是未知、或部分已知；有關障礙物的形狀、大小和位置，必須透過事後偵查或感測獲得，並且之後也許會再變動的，一般則稱為「動態環境」。在動態環境中，機器人必須不斷地觀察目前環境，以取得有限的資訊，並立即作出決策。因此，區域路徑規劃法的環境資訊可以不斷改變。目前，產、學、研界在全域路徑規劃已獲得許多成果，並發展了許多有效的方法；然而目前全域路徑規劃大部分僅應用於電玩遊戲或導航軟體等非實際環境之應用，對於實際的移動機器人或自動車而言，則難以使用全域路徑規劃於瞬息萬變的實際動態環境中。故難度較高的動態環境問題，則成為近年來越來越受到重視的研究方向。

1.2 文獻回顧

一般來說，對於靜態環境問題，路徑規劃是屬於一次性演算法，可找到最短或最佳路徑。最有名的方法是 Dijkstra 演算法[2]與其衍生出來的 A*算法[3]-[5]。Dijkstra 演算法為 1959 年由 Edsger Dijkstra 所提出，為一種圖形式搜索法。該演算法可以找到最短路徑或最佳路徑，然而 Dijkstra 演算法會探索空間中非常大量的節點，所以 Dijkstra 演算法一般被認為效率不佳。1972 年，Peter Hart 提出了 A*演算法，採用了啟發式預估的設計，所以會優先處理空間中較有探索價值的節點，而忽略一些明顯不合適的節點。因此，A*演算法可以比 Dijkstra 演算法更快速地找到路徑，並且獲得與使用 Dijkstra 演算法所得到的路徑，非常接近的結果(如圖一)。

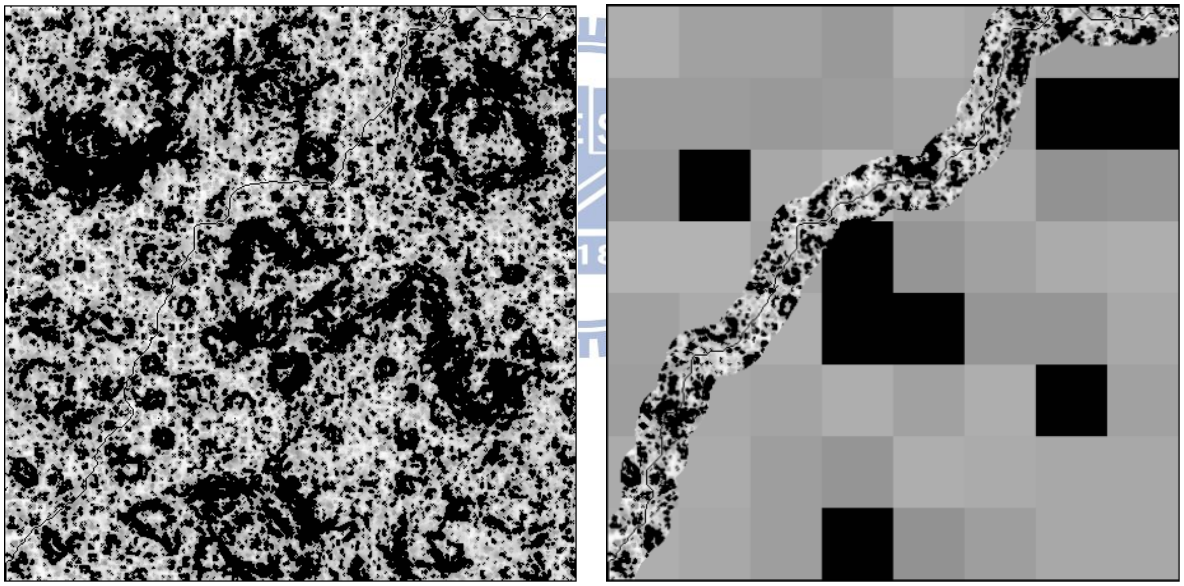
然而對於動態環境而言，像 Dijkstra 與 A*這種一次性搜尋法，是相當沒有效率的。因為當環境發生變化時（例如，出現新的障礙物），Dijkstra 或 A*演算法必須從目前位置重新搜索到終點之路徑。一旦環境變化是持續不斷的，且目前位置離終點仍有很長的距離時，使用 Dijkstra 或 A*演算法將會花費大量的搜索時間，而顯得非常沒有效率。



(a) (b)

圖一、Dijkstra 演算法與 A*演算法之比較圖：(a) Dijkstra 演算法 (b) A*演算法 (來源：Amit's A* Pages[6])

因此，Anthony Stentz 於 1994 年提出了 D*演算法[7][8]。主要為了解決 Dijkstra 和 A*演算法不能處理動態環境的問題。2005 年，Sven Koenig 提出了 D*-Lite 演算法[9]。D*-Lite 演算法比傳統 D *算法更容易被理解，並且擁有更好的效率。其實 D*-Lite 演算法並非由 D *算法發展而來，而是由同樣 Sven Koenig 提出的 LPA*演算法[10] 發展而來。然而，不論是 D*或是 D*-Lite 演算法，它們還是必須一開始做一次全域路徑規劃，所以在第一次搜尋結束前，仍然會有等待時間，特別是針對大型地圖的環境。所以使用 D*演算法於實際大型地圖時，往往要先使用粗略地圖做初期全域路徑規劃(如圖二所示)。

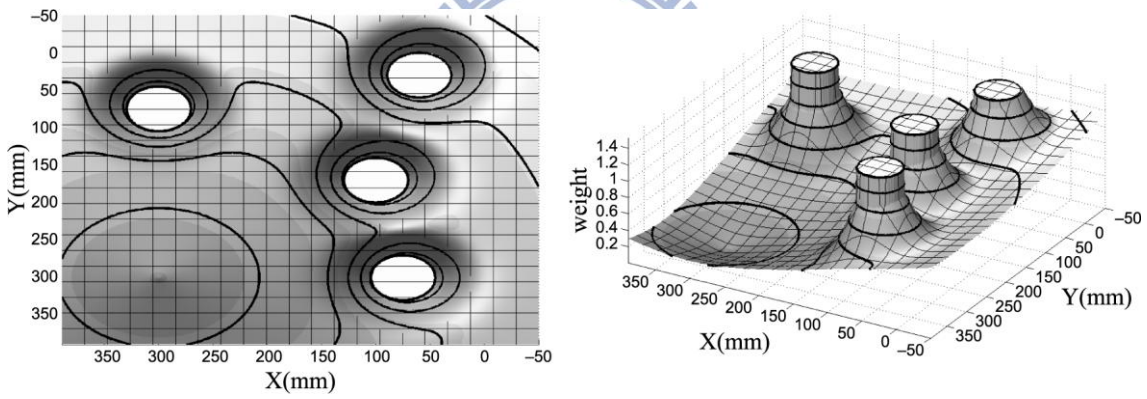


(a) (b)

圖二、D*演算法：(a)直接搜索 (b)先使用粗略地圖進行搜索 (來源：[7])

在過去 20 年，由 Khatib [16]提出的人工位能法，已被廣泛用於移動機器人或機械手臂的路徑規劃問題。起初，人工位能法是設計給機械手臂，以避免碰撞障礙物，並抓取目標物體。但後來研究人員發現，人工位能法也非常適合處理移動機器人的路徑規劃問題，並且可以建立一條

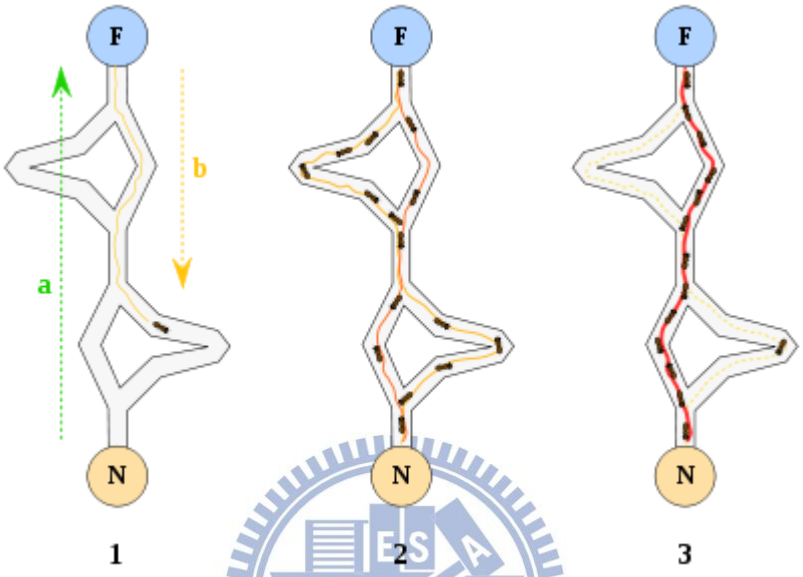
非常平滑的軌跡路徑。雖然人工位能法的理論很簡單，可以即時地處理動態環境問題；但是，人工位能法的缺點也非常顯而易見。例如，最大的缺點就是區域解問題。這是因為當終點產生的引力與障礙物產生的斥力之合力為零時，機器人將停住而無法繼續前進。雖然有許多方法來解決這個問題，例如，添加一個隨機擾動力量，讓移動機器人離開區域解；或是結合其他路徑搜尋方法[20][21]，幫助機器人離開區域解之地區。這些方法或多或少可降低機器人產生區域解的問題。但當障礙物的尺寸與形狀非常龐大與複雜時，這些方法便顯得十分沒有效率。除此之外，人工位能法還有其他問題，例如在狹窄的通道會震盪、或是無法通過更窄的通道。在參考文獻[18][19]有更詳細的描述。



圖三、人工位能法之範例(來源：[17])

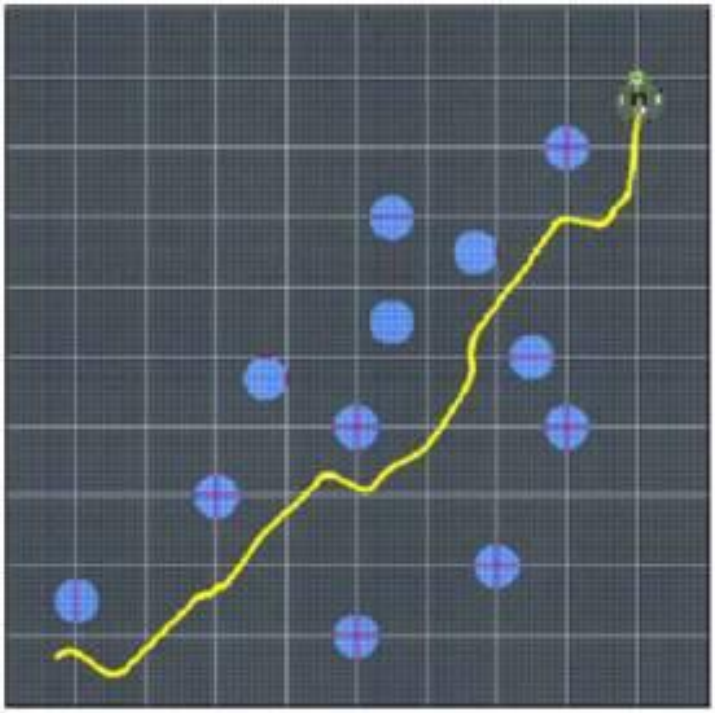
另外還有如蟻群演算法[22]-[25]等最佳化方法，也是近年來常被來應用在路徑規劃上的一種方法。它由 Marco Dorigo 於 1992 年提出，其靈感來源於螞蟻在尋找食物過程中發現路徑的行為。自然界螞蟻出外覓食時，會在行經的路徑上留下一種叫費洛蒙 (Pheromone)的物質，以利後續出發的螞蟻決定是否依循此路徑前進。覓食初期蟻群會漫無目的地隨機四處移動，並留下費洛蒙。路徑愈短，螞蟻會較快地往返巢穴與覓食處，所留下費洛蒙濃度愈高。則較短的路徑將會吸引更多螞蟻循此路覓食，

最後所有螞蟻均循該路線覓食（如圖四所示）。不過蟻群演算法需要較多的計算時間來求得路徑解，且也未能如 Dijkstra 演算法等全域搜尋法能保證求得最短路徑。故蟻群演算法並不普遍應用在機器人的路徑規劃問題上。

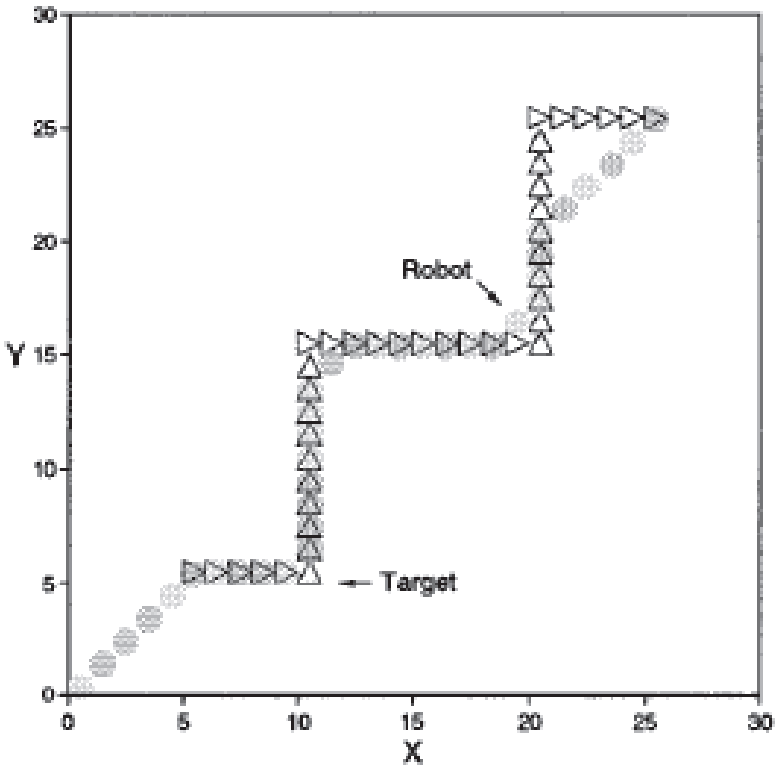


圖四、蟻群演算法之範例（來源： [26]）

其他人工智慧方法[27]，如遺傳算法[28]-[32]、類神經網路系統[33]-[38]，模糊演算法[39][40]...等等。這些方法大部分著重於機器人面對當前障礙物的因應對策，因此對於解決動態環境的即時避碰問題，有相當不錯的效率與成果。然而，這些方法大多未對整個環境做實際的路境規劃，所以當碰到比較複雜的地形（如迷宮式的地圖），這些方法往往產生不盡理想的結果，甚至難以求得路徑解。例如圖五為使用模糊遺傳演算法導引機器人避開障礙物，而這些障礙物均為簡單圓柱體，因此機器人只要繞過障礙物並繼續往終點方向前進即可，並無區域解的問題，而圖六為使用類神經網路系統，控制機器人追逐目標物，此類問題也並未實際執行路徑規劃的動作。



圖五、模糊演算法之範例(來源：[40])



圖六、類神經網路系統之範例(來源：[37])

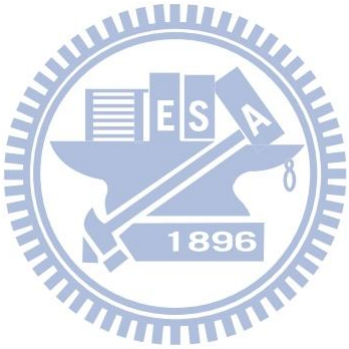
1.3 研究目的

對於移動機器人，有一些現實狀況中會遇到的問題，必須加以解決的。(1)若移動機器人可以在一個靜態、並事先獲得所有資訊的環境中，進行路徑規劃，當然是最理想的狀況。然而，在大多數現實情況下，在每次路徑規劃前，是很難獲得所有的環境資訊的。特別對於大型環境而言，收集環境資訊和建立其模型是非常艱難與複雜的工作。(2) 即使移動機器人在路徑規劃前，已獲得的所有環境資訊。但是現實中環境資訊通常一直在改變(如:行人或行車)。因此，移動機器人必須有能力即時地處理未知和動態環境中，否則很容易導致事故和災害。

故本研究期望開發一套新演算法，能整合先前路徑規劃法之優缺點，以解決實際狀況中，移動機器人所會碰到的種種問題。因此本研究開發之演算法不僅需能解決大型空間、複雜地形(迷宮)、動態環境等等目前路徑規劃法所難以解決的狀況；另外此演算法還需具有線上計算、即時反應的特性，對於應用在真實的移動機器人上，才有較高的實用性。

1.4 論文架構

本論文架構總共分為五章，第一章為緒論，說明研究之背景、動機以及目的。第二章詳細說明古典的 Dijkstra 演算法，與本研究提出的 D++演算法之演算流程。在第二章中還有將 D++與幾個重要的路徑規劃法做比較，並探討其優缺點。在第二章的最後則是應用 D++演算法在舉重機器手臂之模擬實驗。第三章則介紹如何將 D++演算法與實際的移動機器人進行軟硬體的整合，與介紹本研究移動機器人之硬體架構。第四章為展示 D++演算法應用在移動機器人的實驗結果，以驗證 D++演算法具有很高的實用性。最後，第五章為本論文之結論與建議，本章討論 D++演算法在實際應用中的心得，並提出未來需要改善的建議。



二、D++演算法

本研究所提出的新型演算法之所以命名為D++演算法，是因為該演算法乃改良原有的Dijkstra演算法而來的。改良後的D++演算法對於未知與動態環境問題，可以獲得良好的結果。在本章節中，我們將具體描述D++演算法的觀念，與其演算過程。並在本章節的末段展示使用D++演算法的模擬過程與結果 。

2.1 Dijkstra 演算法

由於本研究提出的D++演算法是由Dijkstra演算法改良而成的，故本節中將先介紹Dijkstra演算法的基本理論。Dijkstra演算法是由Edsger Wybe Dijkstra[2]於1959年提出的一個最短路徑演算法，也是目前公認求解最短路徑的高效經典演算法之一。例如Cheng等人[42]曾採用Dijkstra演算法，求解規劃任意三次元曲面兩點間最短路徑的問題。Dijkstra演算法可以保證求出某一節點到其他所有節點的最短路徑。目前最常用的最短路徑演算法－A*演算法，其實也是由Dijkstra演算法衍生而來。Dijkstra演算法與A*演算法的優缺點在於：Dijkstra演算法具有最佳答案的保證性，但搜尋的空間與時間複雜度卻很大；反之，A*演算法的空間與時間複雜度相對於Dijkstra演算法要小很多，但卻不能保證搜尋出的答案為最佳的。

Dijkstra 演算法的主要特點是以起始點為中心向外一層一層擴展，直到擴展到終點為止，所以 Dijkstra 演算法能夠求得最短路徑的最佳答案。Dijkstra 演算法一般對於節點的紀錄有兩種方式，一種用永久和臨時標號方式，一種是用 OPEN、CLOSE 表方式，在本篇論文中採用 OPEN、CLOSE 表的方式。其搜尋過程簡介如下所述：

1. 建立兩個節點紀錄－OPEN 與 CLOSE。OPEN 表紀錄所有已生成而未選取過的節點，CLOSED 表中記錄已選取過的節點。
2. 計算起點的目標函數值，並把它放入 OPEN 表中等待檢查。
3. 從 OPEN 表中找出目標函數最小的節點，找出這個節點的所有子節點，並計算這些子節點的目標函數值。
   A. 若子節點已經在 OPEN 表中了，比較子節點新舊目標函數值，保留目標函數值較小的子節點資料。
   B. 若子節點已經在 CLOSE 表中了，比較子節點新舊目標函數值。若子節點的新目標函數值較小，更新子節點資料並重新加入 OPEN 表中。
   C. 不在 OPEN 表也不在 CLOSE 表中，則直接將子節點加入 OPEN 表中。
4. 重複第 3 步。直到挑選的節點為終點則完成搜尋；或是 OPEN 表裡沒有節點則表示沒有路徑可以到終點。

上述的節點資料包括：(1)節點的座標(2)節點的父節點座標：因為等到搜尋結束時，可以從終點回溯父節點到起點以得到路徑。(3)節點的目標函數值：每個節點的目標函數值為從父節點累加，所以當父節點改變時，節點可能會有不一樣的目標函數值。下面我們將舉一個簡單的例子示範 Dijkstra 演算法。如圖七為一個 3×3 的 2D 網格圖，每個網格到鄰近網格的距離均為 1。欲求 S 點到 G 點的最短路徑，以 Dijkstra 演算法求解的步驟如下。在下列參考範例中的(A_a,B)，A 表示節點編號，a 表示 A 的父節點編號，B 表示節點 A 目前的目標函數值：

S	**1**	**4**
2	**3**	**6**
5	**7**	**G**

圖七、使用 Dijkstra 演算法的 3×3 地圖

1. 挑選：(S,0)

   子節點：$(1_S,1),(2_S,1),(3_S,1)$

   OPEN 表：$(1_S,1),(2_S,1),(3_S,1)$

   CLOSE 表：(S,0)

2. 挑選：$(1_S,1)$

   子節點：$(S_1,1),(2_1,2),(3_1,2),(4_1,2),(6_1,2)$

   OPEN 表：$(2_S,1),(3_S,1),(4_1,2),(6_1,2)$

   CLOSE 表：$(S,0),(1_S,1)$

3. 挑選：$(2_S,1)$

   子節點：$(S_2,2),(1_2,2),(3_2,2),(5_2,2),(7_2,2)$

   OPEN 表：$(3_S,1),(4_1,2),(5_2,2),(6_1,2),(7_2,2)$

   CLOSE 表：$(S,0),(1_S,1),(2_S,1)$

4. 挑選：$(3_S,1)$

   子節點：$(S_3,2),(1_3,2),(2_3,2),(4_3,2),(5_3,2),(6_3,2),(7_3,2),(G_3,2)$

   OPEN 表：$(4_1,2),(5_2,2),(6_1,2),(7_2,2),(G_3,2)$

CLOSE 表：(S,0),(1_S,1),(2_S,1),(3_S,1)

5. 挑選：(4_1,2)

   子節點：(1_4,3),(3_4,3),(6_4,3)

   OPEN 表：(5_2,2),(6_1,2),(7_2,2),(G_3,2)

   CLOSE 表：(S,0),(1_S,1),(2_S,1),(3_S,1),(4_1,2)

6. 挑選：(5_2,2)

   子節點：(2_5,3),(3_5,3),(7_5,3)

   OPEN 表：(6_1,2),(7_2,2),(G_3,2)

   CLOSE 表：(S,0),(1_S,1),(2_S,1),(3_S,1),(4_1,2),(5_2,2)

7. 挑選：(6_1,2)

   子節點：(1_6,3), (3_6,3), (4_6,3), (7_6,3),(G_6,3)

   OPEN 表：(7_2,2),(G_3,2)

   CLOSE 表：(S,0),(1_S,1),(2_S,1),(3_S,1),(4_1,2),(5_2,2),(6_1,2)

8. 挑選：(7_2,2)

   子節點：,(2_7,3),(3_7,3),(5_7,3),(6_7,3),(G_7,3)

   OPEN 表：(G_3,2)

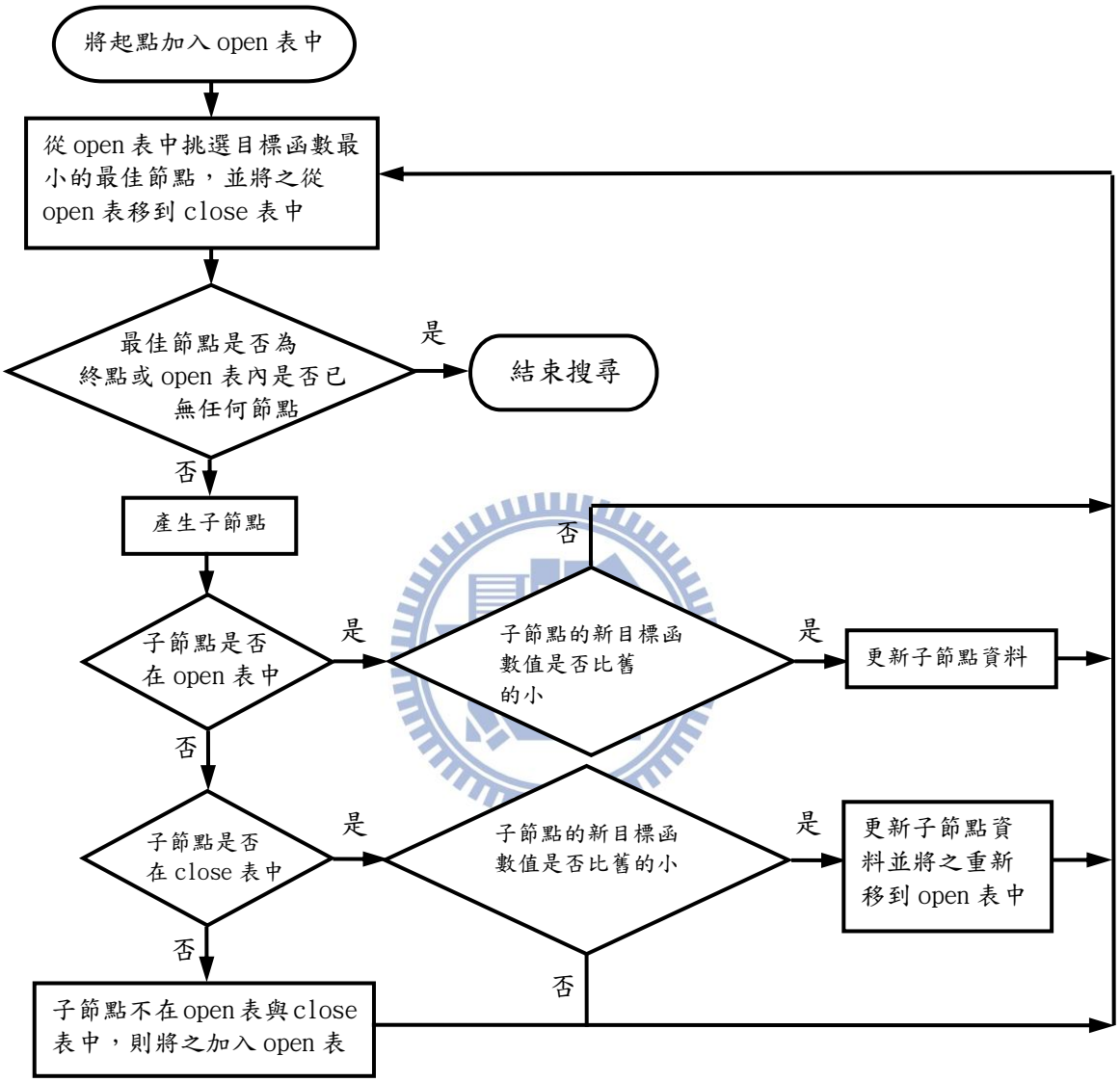
   CLOSE 表：(S,0),(1_S,1),(2_S,1),(3_S,1),(4_1,2),(5_2,2),(6_1,2),(7_2,2)

9. 挑選：(G_3,2) =>挑選到終點，則搜尋結束。

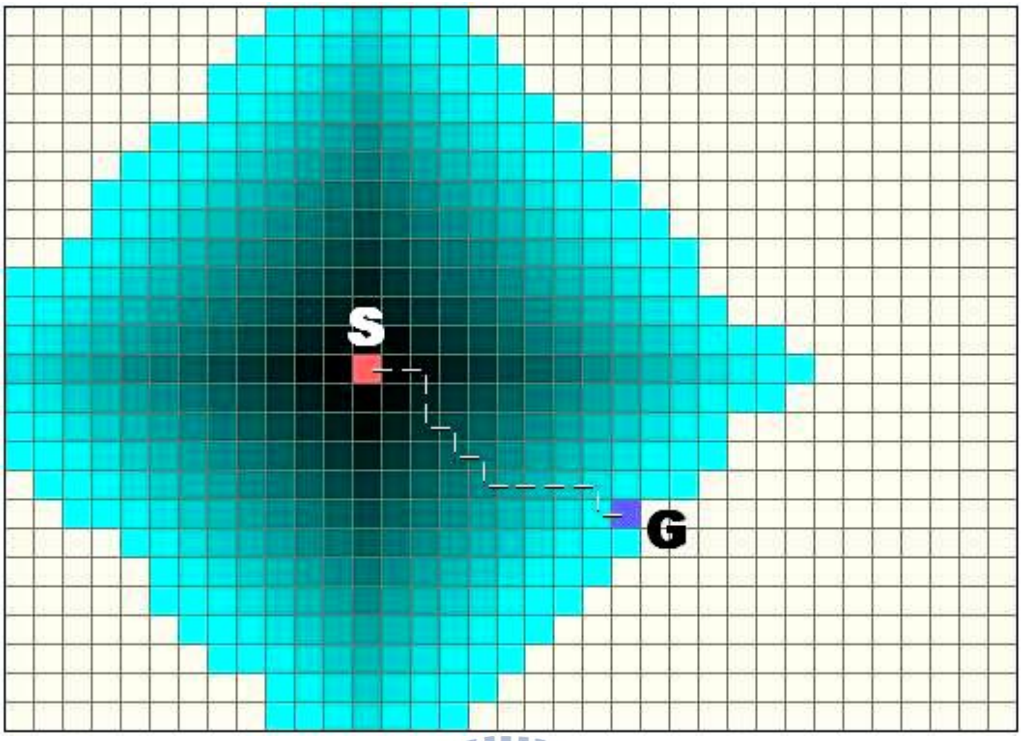
10. 從終點回溯各節點的父節點直到起點為止：(G_3,2)->(3_S,1)-> (S,0)則搜尋結果之最短路徑：S->3->G

有關 Dijkstra 演算法的詳細搜尋流程，可參考圖八所示。接著我們以兩個 2D 平面空間最短路徑規劃的例子，來展示 Dijkstra 演算法的成果，如下面圖九與圖十所示。其中圖九為無障礙範例，而圖十為具障礙物範

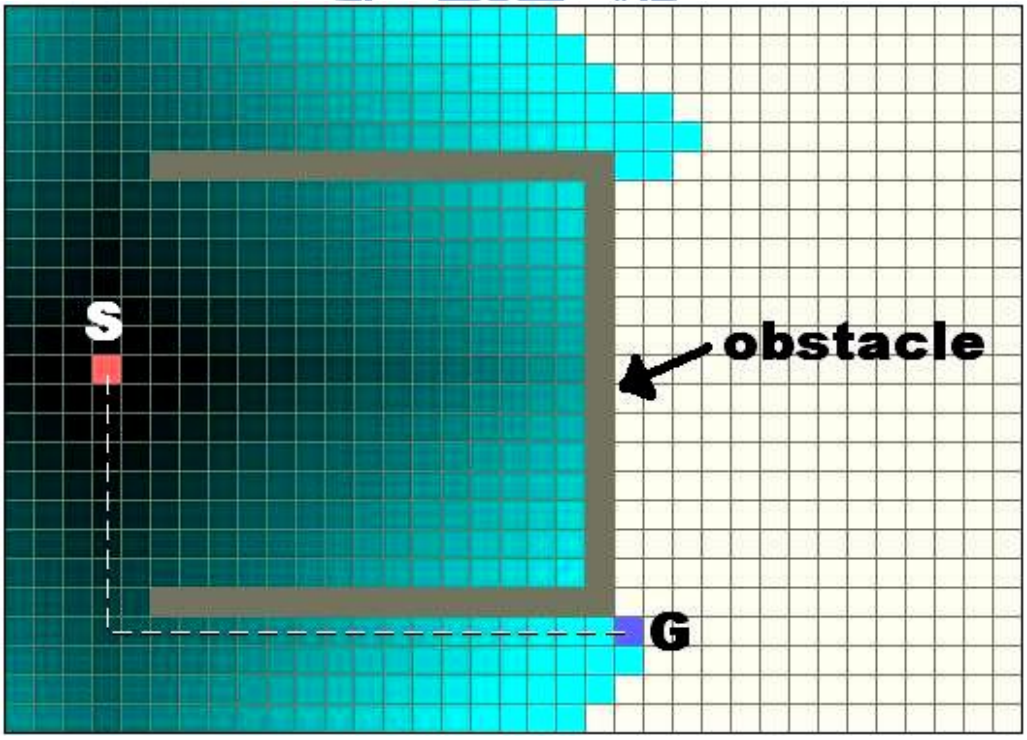
例。在圖中有顏色的格子為在搜尋中，有被討論到的格子；而顏色越深者表示目標函數值越小、離起點越近，反之則越大。



圖八、Dijkstra 演算法流程圖



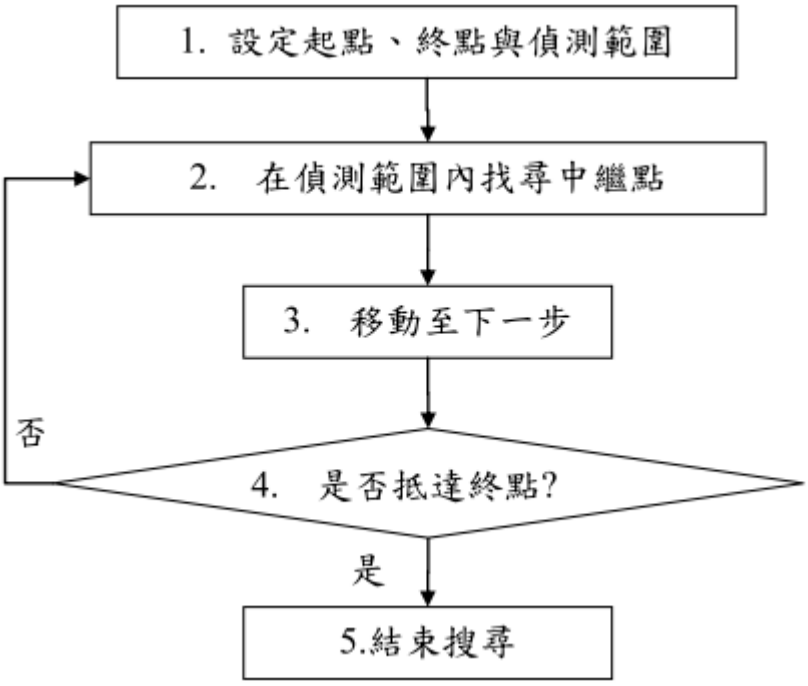
圖九、Dijkstra 演算法無障礙範例
(來源：Amit's A* Pages[6])



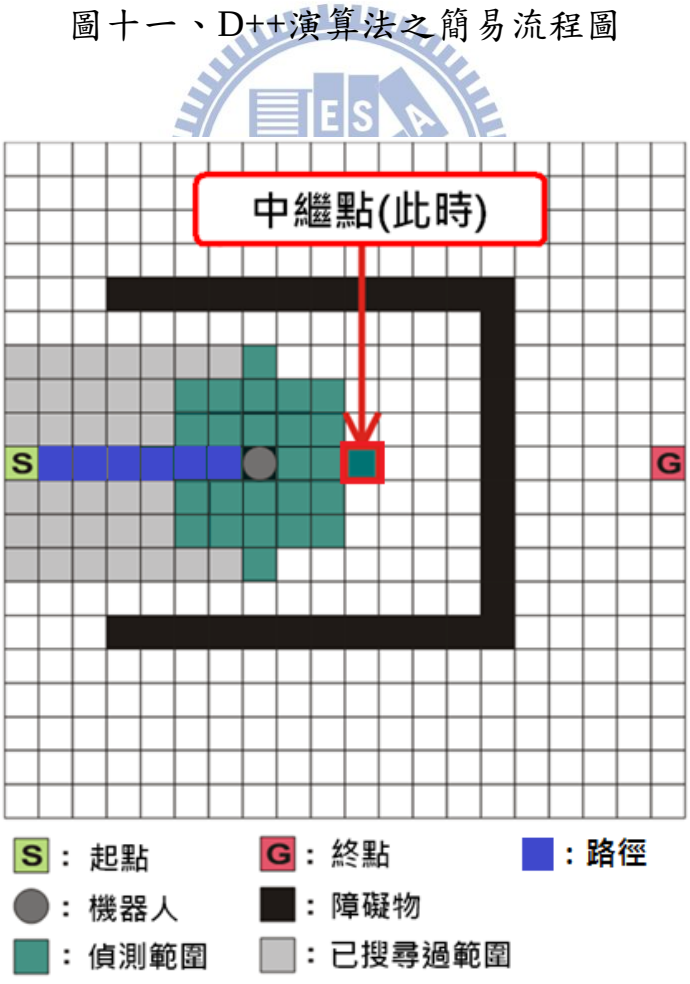
圖十、Dijkstra 演算法具障礙物範例
(來源：Amit's A* Pages[6])

2.2 D++演算法之理論

由於Dijkstra演算法屬於一次性搜尋的全域路徑規劃法。因此對於大型的地圖或環境，Dijkstra演算法需要許多搜尋時間去計算最短路徑，才能令機器人開始移動。另外Dijkstra演算法還需要事先獲得環境中所有的資訊，所以它無法在未知的環境中使用。為了克服這些限制，我們參考了如使用超音波感測器的環境感測路徑規劃方法[43]-[45]，在Dijkstra演算法中加入了的「偵測範圍」概念；如此可使原來的Dijkstra演算法只需搜尋區域環境即可[46]。D++演算法中的偵測範圍類似於一個被感測器偵測到的區域，每次只需讓機器人在偵測範圍中搜尋一個最接近終點的中繼點，並相中繼點前進。僅搜尋偵測範圍內的中繼點可讓機器人在極短的時間內即可決定下一步移動的方向與位置。透過這樣的方法，機器人可以一步一步地逐漸靠近並抵達終點。圖十一描述了D++演算法的概念流程圖，而圖十二則說明了本篇論文中所用到的圖例。如圖十一中之概念流程所示，流程中每次循環迴圈的計算時間將會非常短，這是因為每次搜尋的範圍(即偵測範圍)為有限的小型空間。因此機器人可迅速第一次完成搜尋並開始移動，而無原本Dijkstra演算法常有的等待時間。另外，機器人只需附近的環境資訊即可開始搜尋路徑，而不需要事先取得環境中所有的資訊。



圖十一、D++演算法之簡易流程圖



圖十二、本研究所用到之圖例說明

在圖十一的D++演算法簡易流程中，其步驟二包含了兩個主要執行函式：(1) GET_DETECTIVE_RANGE() 和 (2) GET_NEXT_MOVE()。其中第一個函式 GET_DETECTIVE_RANGE()的執行目的為使用Dijkstra演算法，從目前位置不停擴張子節點範圍到所設定的偵測範圍大小。至於第二個執行函式GET_NEXT_MOVE()則是搜尋偵測範圍內，最接近終點的中繼點並獲得從目前位置到中繼點的路徑，進而獲得下一步移動的方向與位置。執行函式GET_DETECTIVE_RANGE() 的程式條列如下：

Function: *GET_DETECTIVE_RANGE()*

L1 **for**(;;)
L2 *X* = *GET_BEST_NODE()*
L3 **if** (*X* is *GOAL*) **then**
L4 break
L5 **if** (*X* is *NEW*) **then**
L6 add *X* into *SELECT_LIST*
L7 **if** (*COST_X* > *DETECTIVE_RANGE*) **then**
L8 **if** (*SELECT_LIST* is not empty) then
L9 break
L10 **else**
L11 set *X* as *OLD*
L12 remove *X* from *OPEN_LIST*
L13 add *X* into *CLOSE_LIST*
L14 **for**(;;)
L15 *Y* = *GET_CHILD_NODE(X)*
L16 **if** (*Y* is in *OPEN_LIST* & *NEW_COST_Y* < *OLD_COST_Y*) **then**
L17 refresh data base of *Y*

L18 **else if** (*Y* is in *CLOSE_LIST* & *NEW_COST_Y* < *OLD_COST_Y*) **then**

L19 refresh data base of *Y*

L20 remove *Y* from *CLOSE_LIST*

L21 add *Y* into *OPEN_LIST*

L22 **else if** (*Y* is not in *OPEN_LIST* & *CLOSE_LIST*) **then**

L23 write data base of *Y*

L24 add *Y* into *OPEN_LIST*

L25 **if** (X has no other child node) **then**

L26 Break

在執行函式GET_DETECTIVE_RANGE()中還包括了兩個子執行函式：GET_BEST_NODE()與GET_CHILD_NODE()。其中第一個子執行函式GET_BEST_NODE()為從OPEN_LIST中挑出移動成本最小的節點X。而第二個子執行函式GET_CHILD_NODE()則是取得離節點X最近的各個子節點。 而在執行函式GET_DETECTIVE_RANGE()中的OPEN_LIST，儲存了已被搜尋到但尚未被挑選的節點。而CLOSE_LIST則儲存了已被挑選過的節點。因為執行函式GET_DETECTIVE_RANGE() 在函式一開始即清除OPEN_LIST 和CLOSE_LIST的內容，所以這兩個表在一個搜尋循環後，並無法保留環境中節點曾被拜訪過的紀錄。這樣可能會造成機器人不斷地拜訪某些舊節點，而在某個區域裡打轉。為了避免發生這樣的情形，D++演算法參考了D*演算法[7][8]的方法，以在一個搜尋循環後持續記錄節點被拜訪的情形。因此D++演算法在搜尋一開始(即圖十一中的第一步驟)，即將環境中所有的節點設定為”NEW”的狀態(即為未被拜訪過狀態)。如果節點一旦被偵測或拜訪過後，它的狀態就會被變更為”OLD”。另外D++演算法還使用了SELECT_LIST。在偵測範圍內的節點被搜尋到

時，如果節點的狀態為NEW，則它將被加入到SELECT_LIST中；否則它將被忽略不予討論。之後每次搜尋循環的中繼點則將由位於SELECT_LIST的節點中被挑選。在執行函式GET_DETECTIVE_RANGE()所有在節點之COST指的是從目前機器人的位置移動到目前討論節點的路徑長度，如式(2-1)。

$$\cos t_C = \cos t_L + \sqrt{(X_C - X_L)^2 + (Y_C - Y_L)^2} \tag{2-1}$$

其中$cost_C$ 為經計算後之目前節點的移動成本，$cost_L$表示上一節點的總移動成本，X_C 表示目前節點的X座標，X_L表示上一節點的X座標，Y_C表示目前節點的Y座標，而Y_L 則表示上一節點的Y座標。

至於第二個主要執行函式GET_NEXT_MOVE()的執行程式條列如下表所示：

Function: *GET_NEXT_MOVE()*

L1 **for**(;;)
L2 *X = GET_NODE_FROM_SELECT_LIST()*
L3 *EXPECTED_COST_X = GET_EXPECTED_COST(X)*
L4 **if** (*EXPECTED_COST_X* < *MIN_EXPECTED_COST*) **then**
L5 *WAYPOINT = X*
L6 **if** (*SELECT_LIST* has no other node) **then**
L7 Break
L8 *Y = WAYPOINT*
L9 **for**(;;)
L10 *Z = GET_FATHER_NODE (Y)*
L11 **if** (*Z* = robot's current location) **then**
L12 *NEXT_MOVE = Y*

L13 Break

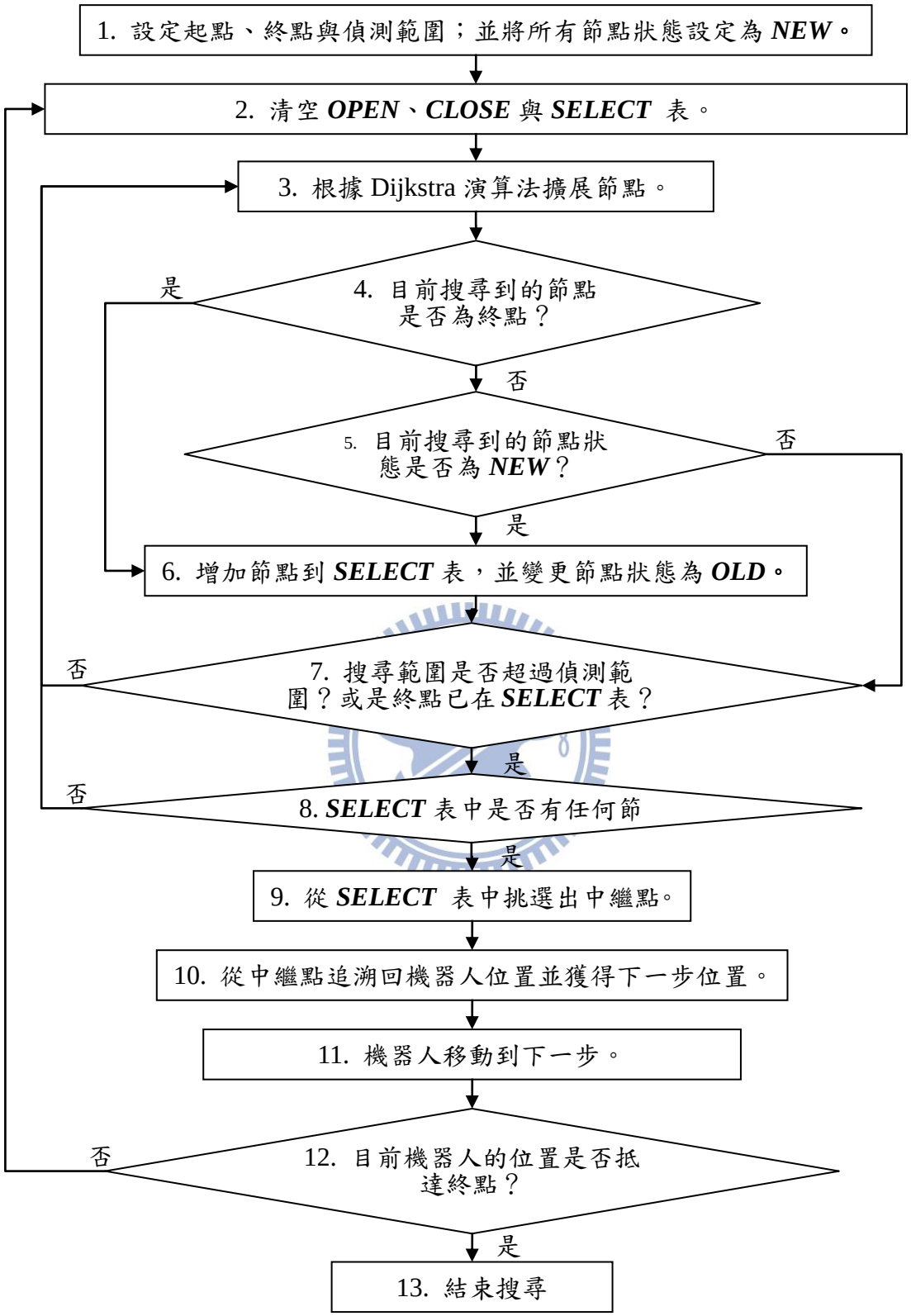
L14 **Else**

L15 $Y = Z$

在執行函式GET_NEXT_MOVE()中，還包含了三個執行子函式，分別為(1) GET_NODE_FROM_SELECT_LIST()、(2) GET_COST()與(3) GET_FATHER_NODE()。其中子函式GET_NODE_FROM_SELECT_LIST()為從SELECT_LIST中取得節點。而GET_EXPECTED_COST()則為計算從節點到終點的預估移動成本，如式(2-2)。

$$\cos t_C = \sqrt{(X_G - X_C)^2 + (Y_G - Y_C)^2} \tag{2-2}$$

其中$cost_C$ 為經計算後、從SELECT_LIST挑選出節點的移動成本，X_C 表示目前節點的X座標，X_G表示終點的X座標，Y_C表示目前節點的Y座標，而Y_G 則表示終點的Y座標。之後在表SELECT_LIST中且離終點GOAL最近的中繼點WAYPOINT便會被挑選出來。透過子函式GET_FATHER_NODE()，D++演算法可由中繼點不斷地回溯各子點的父節點，以獲得從目前機器人的位置到中繼點的路徑。如此機器人也可獲得移動下一步的位置與方向。在一次搜尋循環結束後和下一次搜尋循環開始前，表SELECT_LIST 、表OPEN_LIST和表CLOSE_LIST的內容都將被清除。接著我們總結整個D++演算法的主要流程，並將其繪製如圖十三所示，以俾利更容易了解D++演算法。



圖十三、D++演算法主要流程圖

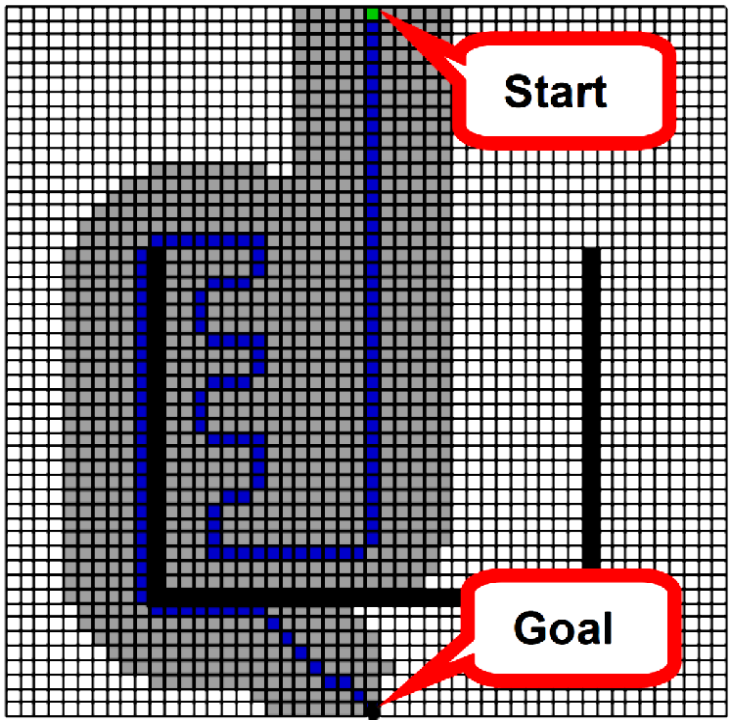
2.3 區域解問題

在本節中，我們將展示一些使用 D++演算法進行路徑規劃可能會碰到區域解的模擬實驗。在這些模擬實驗中，每次的循環週期為 20 毫秒。而在本研究中，所有模擬實驗所使用的電腦體規格則條列於表 1 中：

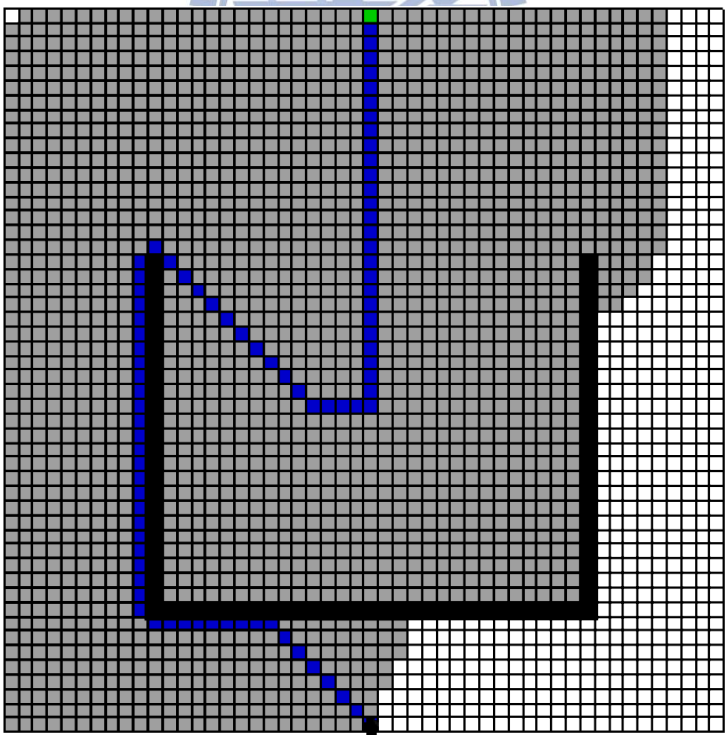
表 1、本研究實驗之電腦硬體規格

Item	Specification
CPU	Intel Core 2 Duo T6600 2.2 GHz
RAM	DDR2-800 2 GB
VGA	ATI Radeon HD4650 512 MB
OS	Microsoft Windows XP
IDE	SharpDevelop

在這個靜態環境路徑規劃的模擬實驗中，將展示兩個例子。其中一個例子的偵測範圍設定為 5 格(如圖十四所示)，而另一個例子的偵測範圍則設定為 20 格(如圖十五所示)。圖十四中路徑的長度明顯比圖十五的路徑還要長的許多，這是因為 5 格的偵測範圍不夠讓機器人提早發現無效的節點與路徑，因此在 U 型障礙物中間遊走了一段路程才脫離出來。相對地，由於 20 格的偵測範圍可即早地預測到 U 型障礙物中間的無效節點與路徑，因此圖十五中，機器人幾乎沒有進入 U 型障礙物便轉換方向移動，如此規劃出來的路徑既短又有效率。

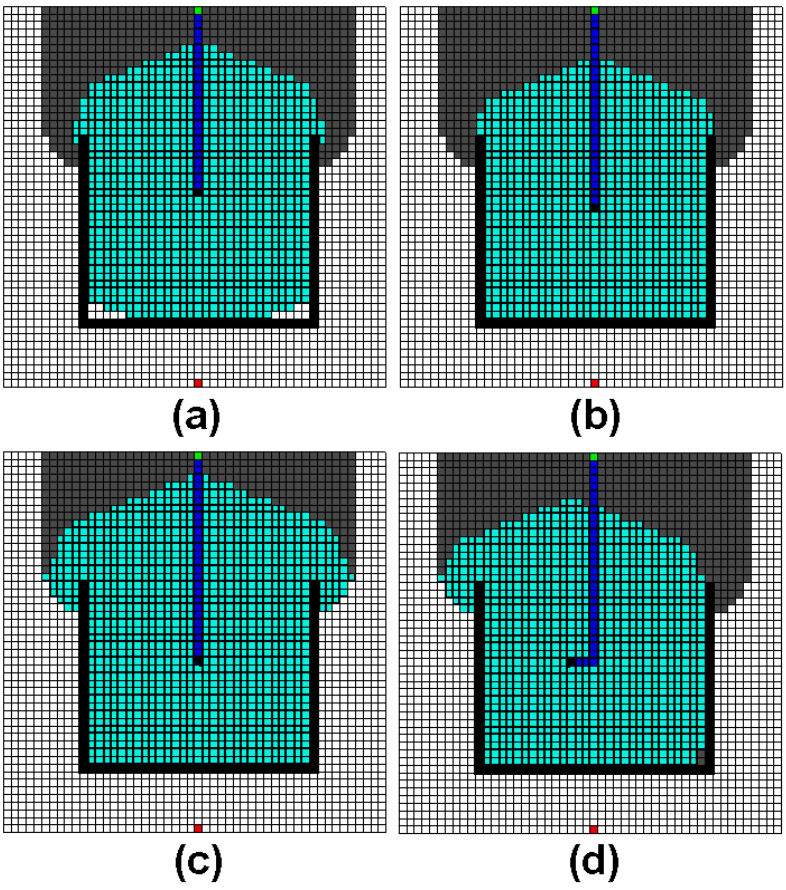


圖十四、偵測範圍為 5 格，總花費時間約 4 秒



圖十五、偵測範圍為 20 格，總花費時間約 3 秒

一般情況下，當一次搜尋循環的搜尋範圍到達了所設定的偵測範圍大小，D++演算法會結束該循環並從表 SELECT_LIST 中挑選中繼點 WAYPOINT。然而，如果機器人進入一條死路中，例如一個 U 型障礙物，則那時的偵測範圍內的節點狀態可能都會是”OLD”(如圖十六 b 所示)。如此表 SELECT_LIST 中將沒有任何節點，而機器人將無法決定如何移動下一步。所以當機器人遇到了這樣的狀況時，D++演算法特別地會擴大偵測範圍直到一個狀態為” NEW”的節點被發現(如圖十六 c 所示)。。透過這樣的方式，機器人可持續規劃到終點的移動路徑並輕易地脫離該區域。

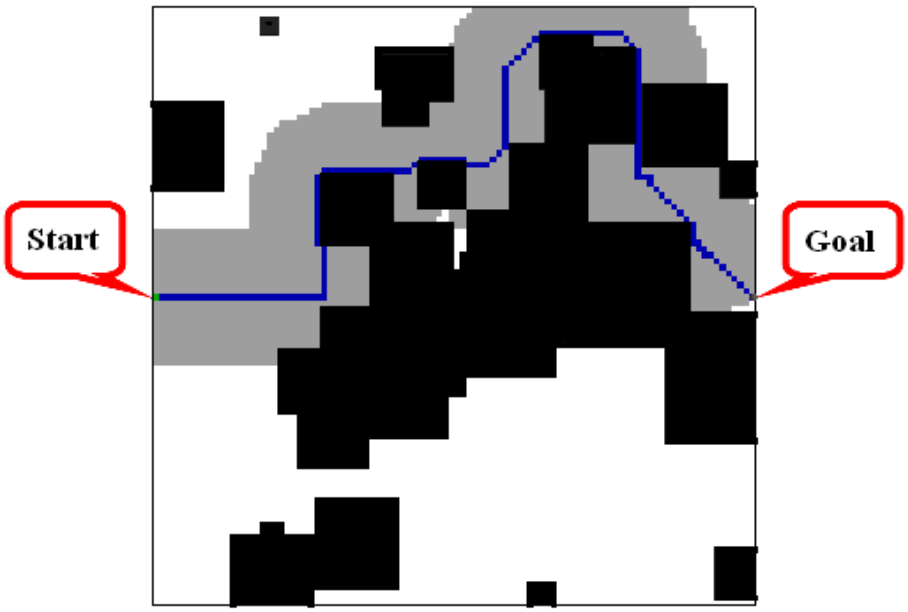


圖十六、當 Select_List 中無節點時，D++演算法的行為模式
(a)機器人進入U型障礙物(b)偵測範圍內沒有狀態為”NEW”的節點(c)D++演算法擴大偵測範圍直到發現有狀態為”NEW”的節點(d)機器人往狀態為”NEW”的節點移動

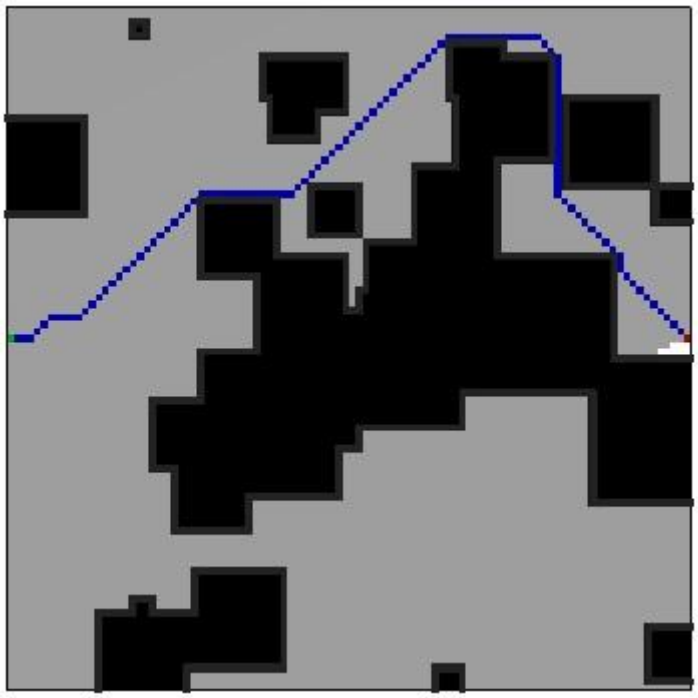
2.4 與其他演算法的比較

(1)靜態環境：

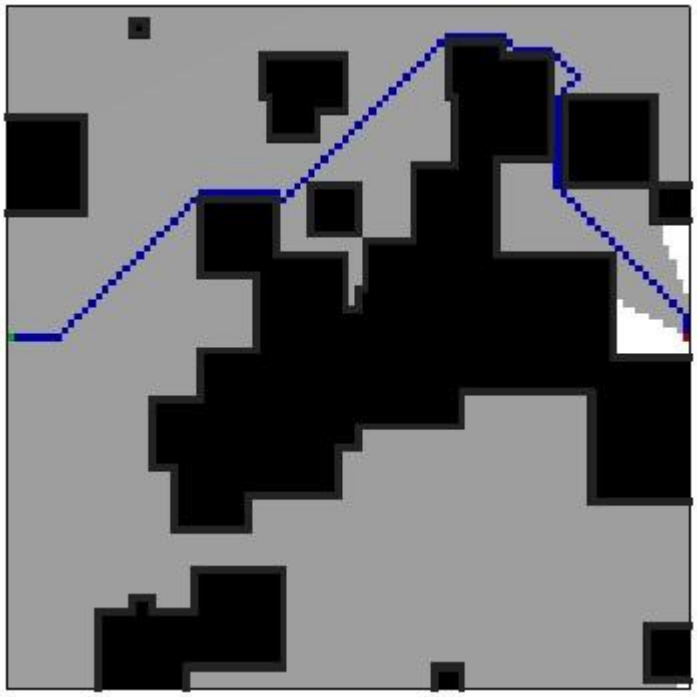
圖十七為一個 100×100 格、且具有模擬野外障礙地形的地圖。起點被設定在地圖的左方，而終點被設定在地圖右方。圖十七為使用 D++演算法規劃路徑後的結果，此模擬實驗使用 15 格為偵測範圍，並以 20 毫秒為搜尋週期，搜尋過程中共探訪了 2626 個格點，最後出來的路徑長度為 151 個格點。圖十八為使用 Dijkstra 演算法規劃路徑後的結果，搜尋過程中共探訪了 6286 個格點，最後出來的路徑長度為 122 個格點，計算搜尋時間約為 2.5 秒。圖十九為使用 A*演算法規劃路徑後的結果，搜尋過程中共探訪了 6209 個格點，最後出來的路徑長度同樣為 122 個格點，計算搜尋時間約為 2.4 秒。圖二十為使用 D*演算法規劃路徑後的結果，因演算概念與 A*演算法相同，故搜尋過程中一樣共探訪了 6209 個格點，最後出來的路徑長度同樣為 122 個格點，計算搜尋時間約為 2.4 秒。最後圖二十一為使用人工位能法，並以 20 毫秒為計算週期之路徑規劃結果，最後卡在地圖中央的區域解，並無法求得路徑解而抵達終點。



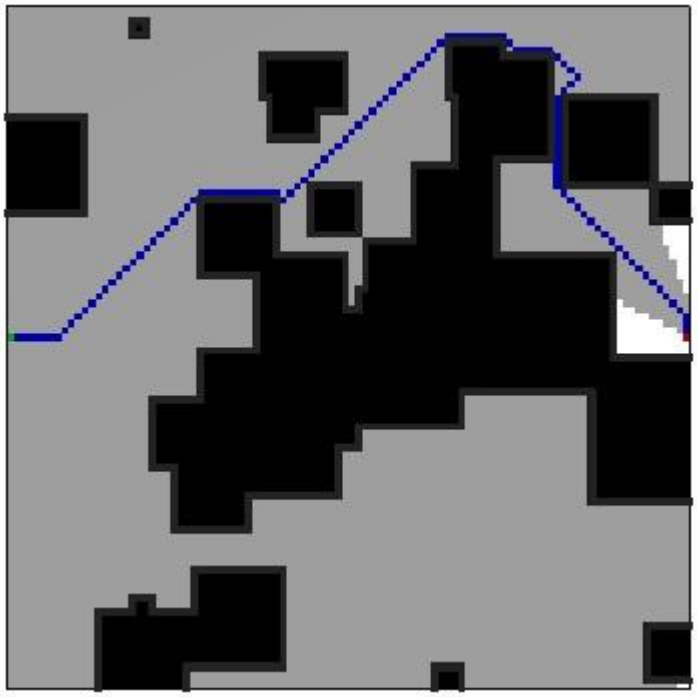
圖十七、靜態環境中使用 D++演算法之路徑規劃結果(探訪節點數:2626, 路徑長度:151)



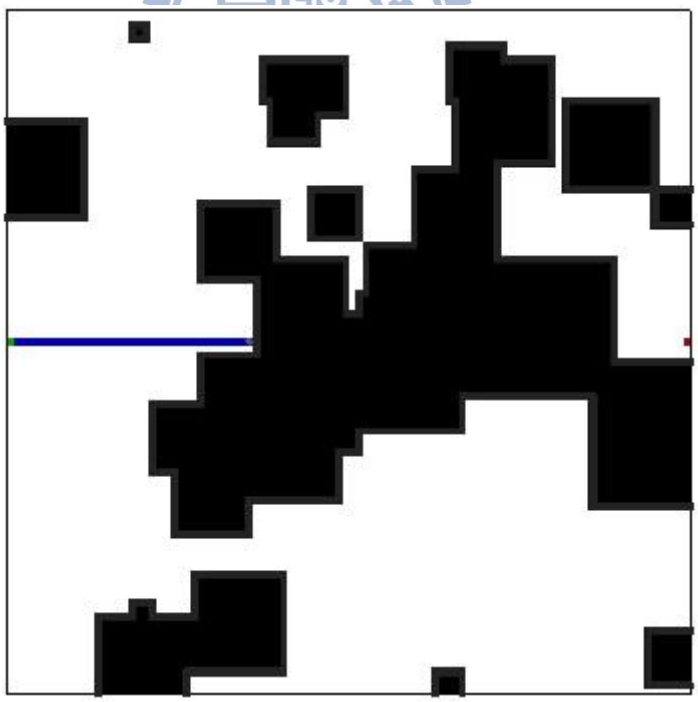
圖十八、靜態環境中使用 Dijkstra 演算法之路徑規劃結果(探訪節點數:6286, 路徑長度:122)



圖十九、靜態環境中使用 A*演算法之路徑規劃結果(探訪節點數:6209, 路徑長度:122)



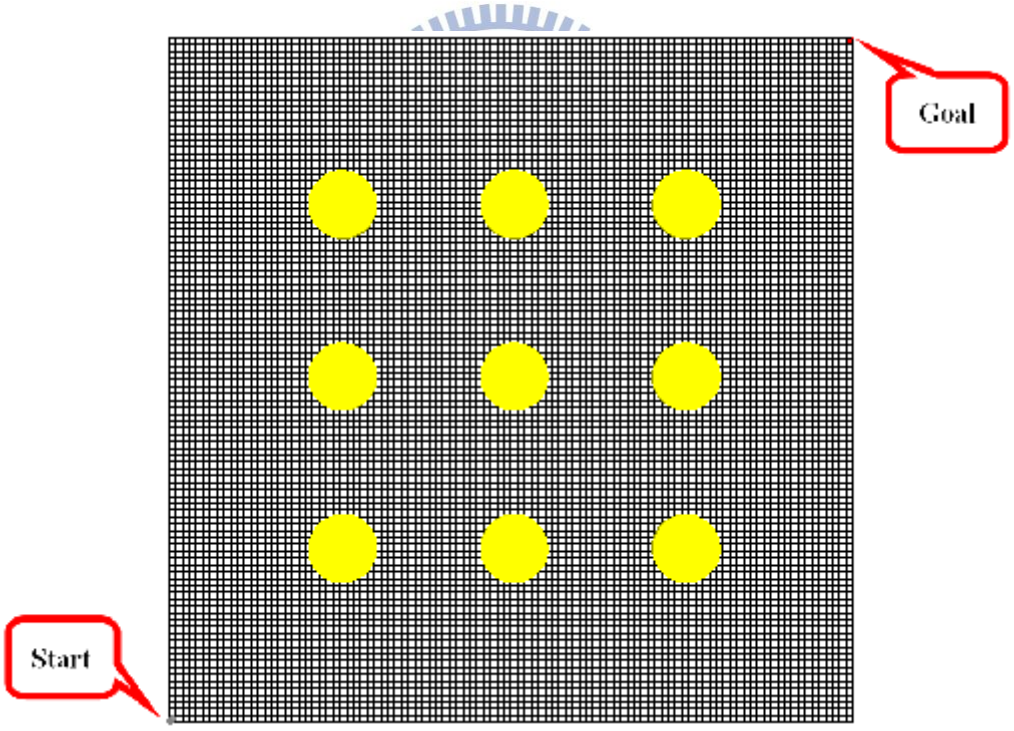
圖二十、靜態環境中使用 D*演算法之路徑規劃結果(探訪節點數: 6209, 路徑長度:122)



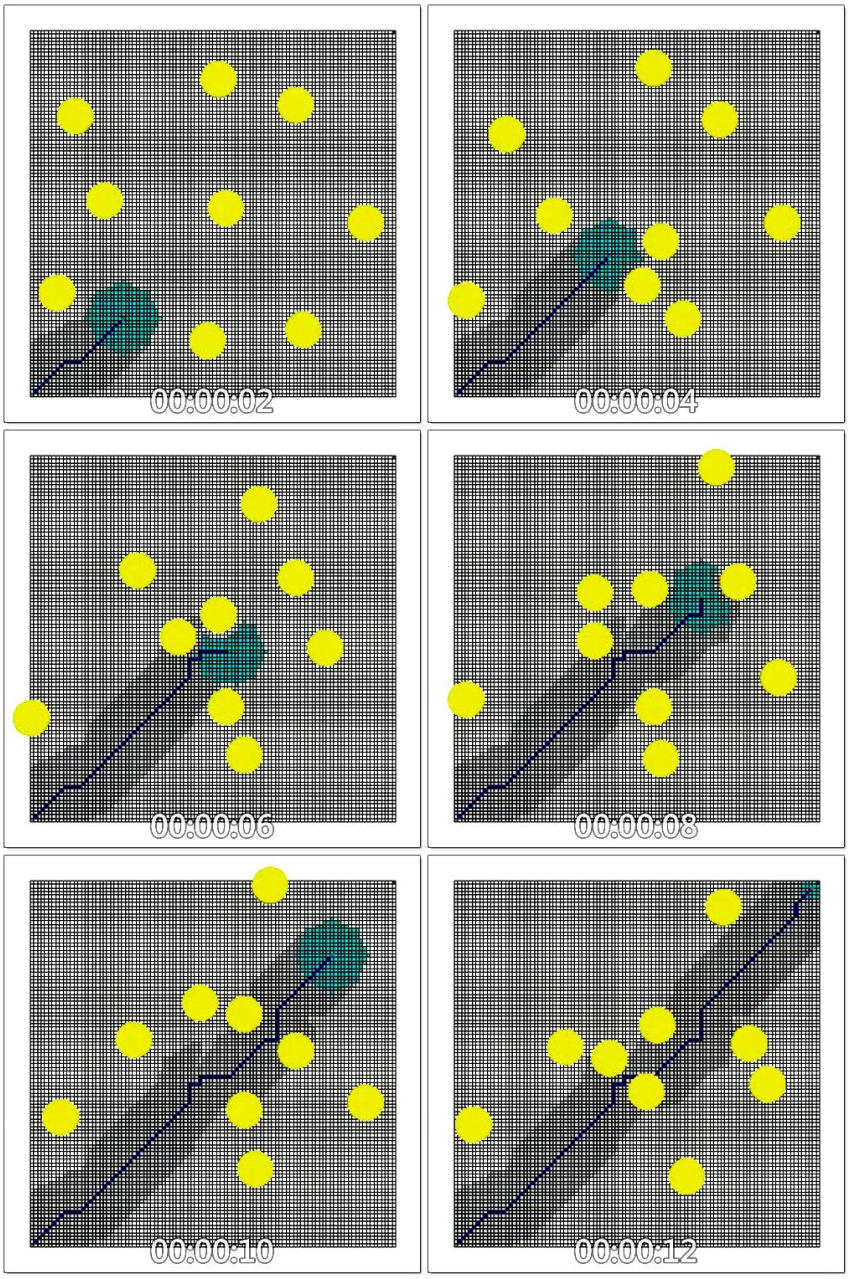
圖二十一、靜態環境中使用人工位能之路徑規劃結果(未能抵達終點)

(2) 動態環境：

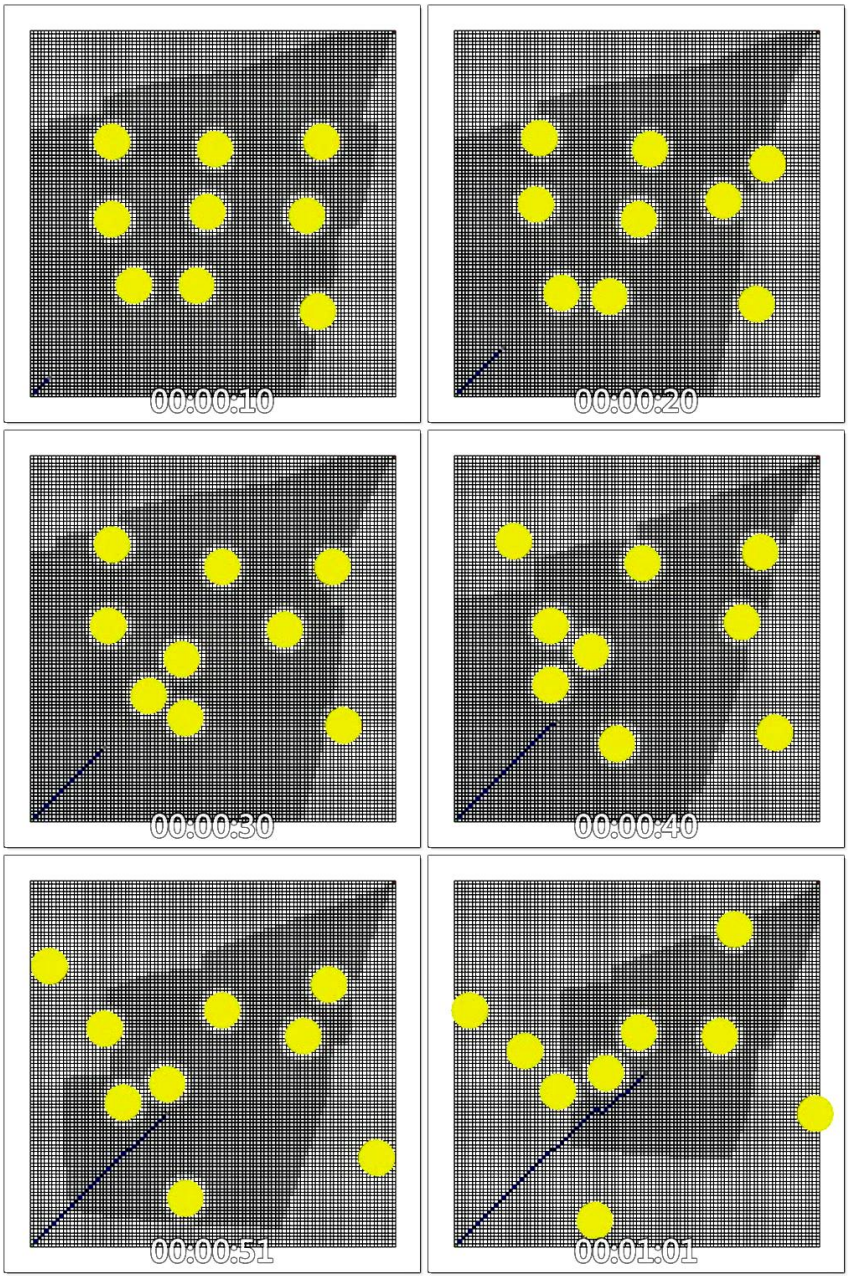
在這個模擬實驗中，我們設計一個 100×100 的大型地圖（如圖二十二所示），一開始在地圖中間有九個圓形物體。這些物體固定以每 100 毫秒的週期，隨機地往或前、或後、或左、或右移動。機器人的起始位置在地圖的左下方，目的地在地圖的右上方。我們以 100 毫秒為計算週期，分別以 D++演算法、D*演算法與人工位能法進行動態路徑規劃，並分別得到如圖二十三、圖二十四與圖二十五之結果。圖二十三中使用 D++演算法共花費 14 秒使機器人從起點移動到終點；圖二十四中使用 D*演算法共花費 1 分 11 秒使機器人從起點移動到終點；圖二十五中使用人工位能法共花費 13 秒使機器人從起點移動到終點。



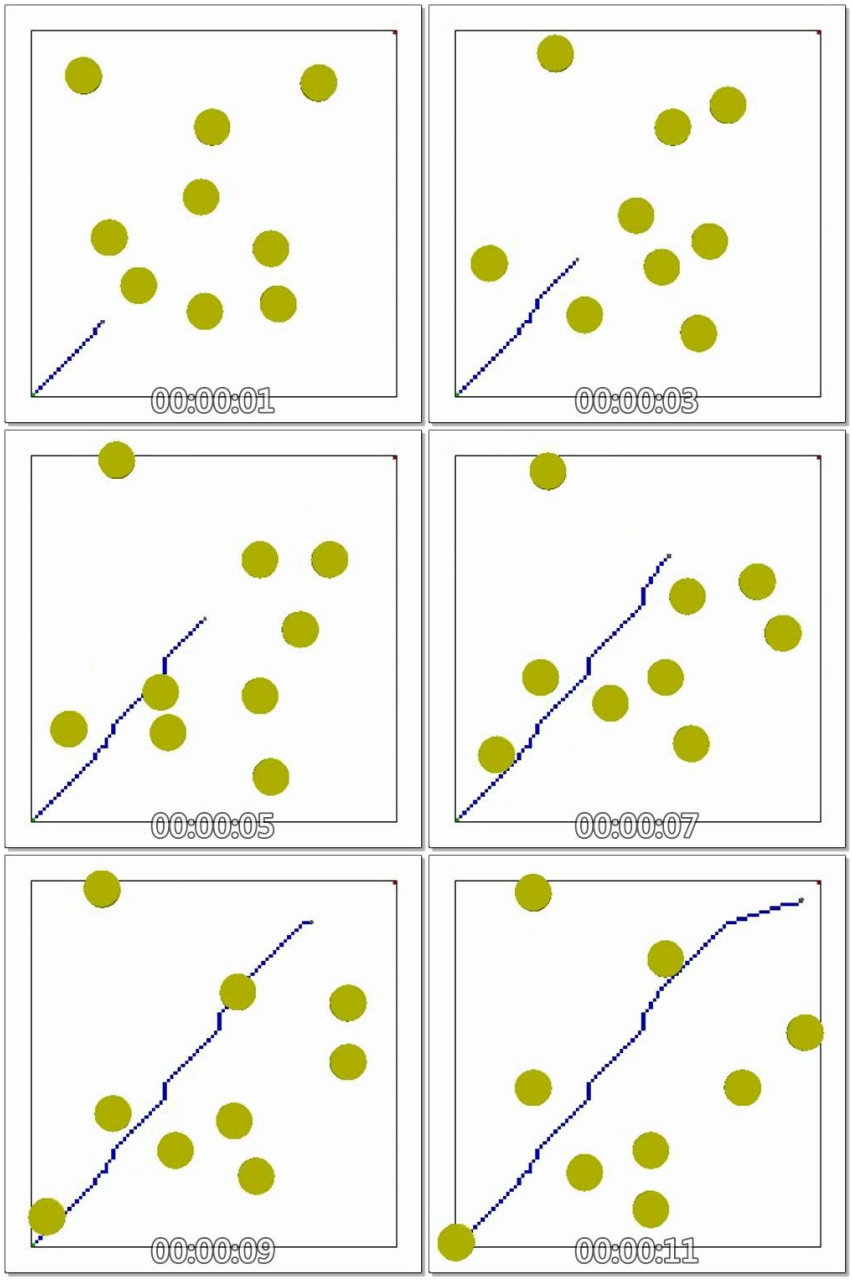
圖二十二、有九個隨機移動物體的動態環境



圖二十三、動態環境以 D++演算法之路徑規劃結果 (偵測範圍為 10 格)



圖二十四、動態環境以 D*演算法之路徑規劃結果



圖二十五、動態環境以人工位能法之路徑規劃結果

(3) 結果比較討論：

我們將前述之靜態環境與動態環境的兩個範例，分別應用了 Dijkstra 演算法、A*演算法、D*演算法、人工位能法和 D++演算法的實驗結果，總結整理如表 2 所示。

表 2、路經規劃結果比較表

	靜態環境				動態環境
	搜尋節點數	路徑長度	機器人啟動時間(秒)	總花費時間(秒)	移動總花費時間(秒)
Dijkstra	6286	122	2.5	124.5	X
A*	6209	122	2.4	124.4	X
D*	6209	122	2.4	124.4	71
人工位能法	X	X	0.02	X	13
D++	2626	151	0.02	154.02	14

從表 2 中我們可以看到在靜態環境的實驗中，人工位能法因為陷入區域解的關係，並沒有搜尋出可到達終點的路徑。而 Dijkstra、A*與 D*則找到比 D++更短的路徑，不過它們也搜尋了比 D++多出一倍的節點數量，因此大約花了 2.5 秒的時間找到路徑解，才可讓機器人開始移動。而 D++雖然求得的路徑稍長，但由於一開始僅做 10 格偵測範圍的區域搜尋，故可順利地在 20 毫秒內完成中繼點的搜尋，並立即讓機器人開始移動。不過若假設機器人移動一個格點需費時 1 秒，並將搜尋時間與機器人移動時間總和，Dijkstra、A*與 D*僅需約 124 秒左右，而 D++則需花費約 154 秒左右。

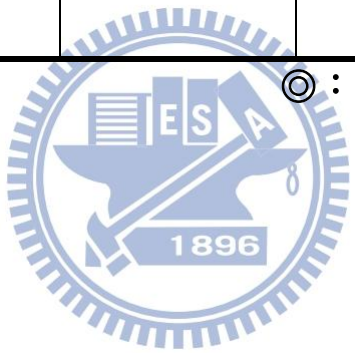
而動態環境中，Dijkstra 與 A*由於屬於一次性搜尋路徑規劃法，故無法處理動態環境的問題。至於 D++與人工位能法則發揮了區域搜尋的優勢，故能觀察動態環境的變化後，便立即快速地做出適當的反應；因此兩者均在 13~14 秒左右，即讓機器人完成在此動態環境從起點移動到終點。而 D*由於在移動的初期具有很大的搜尋範圍，故在此段時期所耗費的路徑規劃時間會較久；而雖然在後段時期因為搜尋範圍變小了而增快了反應速度，然而機器人從起點移動到終點的總花費時間還是高達了 71 秒，為使用 D++與人工位能法的六倍左右。

故從這兩個實驗結果可以得知，Dijkstra 與 A*因為屬於一次性搜尋法，故無法處理動態環境問題；而人工位能法則因為容易遇到區域解問題，故無法處理某些複雜環境問題。而 D*雖然對於動態環境與複雜環境問題均可以處理，但是對於大型地圖仍然有搜尋範圍過大導致搜尋時間過長的問題。特別對於大型動態環境，使用 D*可能因為搜尋時間過久讓機器人停滯，而撞上高速移動的障礙物。因此相較起來，D++比其他方法，雖然分別對於靜態環境與動態環境問題，並非其中最有效率的方法。但 D++可同時適用於靜態環境與動態環境的特性，對於應用在機器人問題上，D++相較於其他方法，則具有較高的泛用性與實用性。綜合以上討論，我們整理各路徑規劃法之優缺點比較於表 3，以俾利更簡單、清楚地了解各方法在各條件下之優勢與劣勢。

表 3、各路經規劃法的優缺點比較

	最短路徑	避免 區域解	小型、靜態、 已知環境	大型、動態、 未知環境
Dijkstra	◎	◎	◎	△
A*	◎	◎	◎	△
D*	◎	◎	◎	○
人工位能法	△	△	○	◎
D++	○	○	○	◎

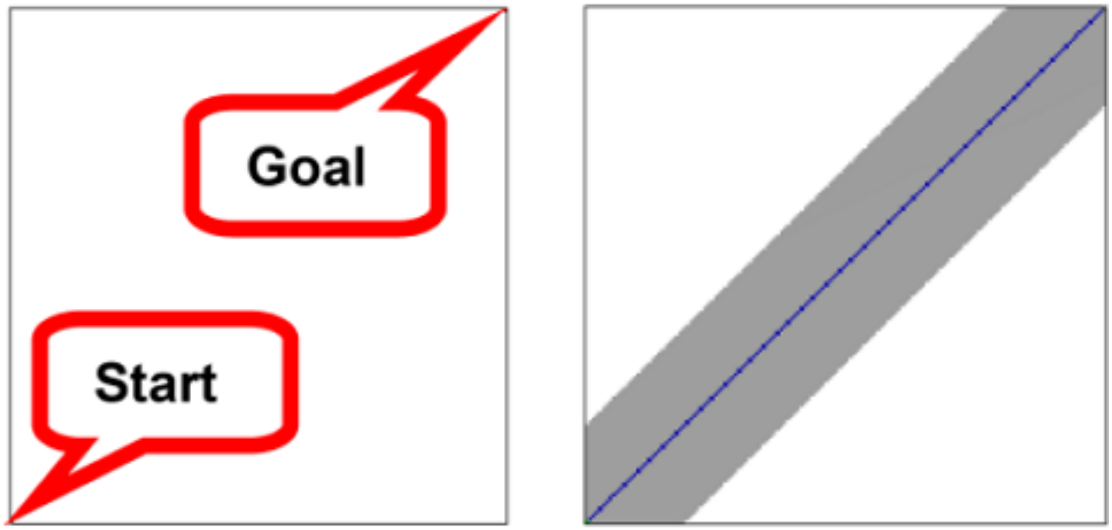
◎：好　　○：普通　　△：劣



2.5 偵測範圍的最佳化設定

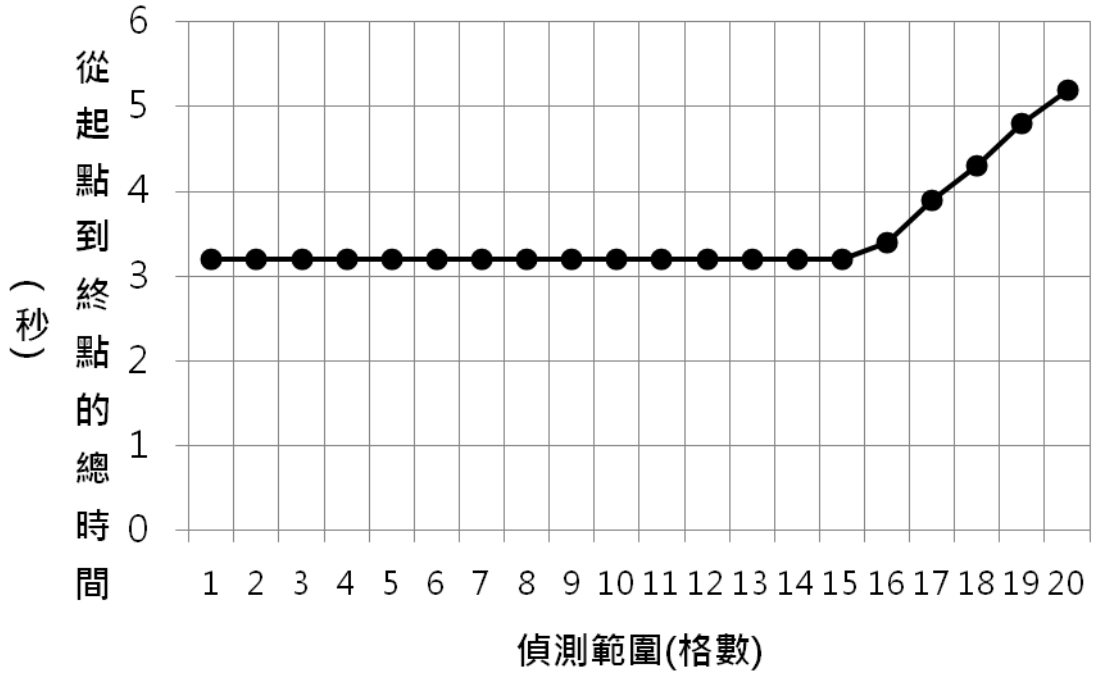
從圖十四與圖十五的結果我們可以得知，偵測範圍的尺寸將影響D++演算法的效能與路徑結果。然而多大的偵測範圍才是適當的?由 D++演算法規劃的路徑可能不是最短、或最佳路徑，這是因為 D++演算法每次搜尋循環僅在區域範圍搜尋中繼點。因此偵測範圍照理說應該儘可能地設定越大越好，只要機器人的控制硬體效能可以在容許的週期時間內完成每次的搜尋循環。例如圖二十六為一個 100×100 格、無障礙物的空白地圖；起點在地圖的左下角，而終點在地圖的右上角。設計無障礙物的地圖的原因是讓不論偵測範圍的大小為多少，使用 D++演算法規劃出的路徑均為從左下角直線移動到右上角。如此我們可以統一路徑的總長度。

接著我們設定偵測範圍從 1 格到 20 格各使用 D++演算法跑一次模擬結果，並設定以 20 毫秒為每次的搜尋循環週期。在此 20 次模擬實驗中，路徑規劃與機器人移動的總時間花費統計如圖二十七所示。透過該圖表的顯示，我們可以看出當偵測範圍小於或等於 15 格時，偵測範圍變大並不會影響路徑規劃與機器人移動的總時間花費；而當偵測範圍大於 15 格時，路徑規劃與機器人移動的總時間花費隨著偵測範圍變大而增加。因此我們可以推斷：對於此機器人的控制硬體而言，若要以每次 20 毫秒的循環週期內完成對偵測範圍大小的環境搜尋中繼點，則對此機器人的控制硬體效能而言，我們將偵測範圍設定為 15 格，將獲得最佳的搜尋效率與路徑結果。



(a) (b)

圖二十六、一個 100×100 格、無障礙物的空白地圖



圖二十七、使用不同偵測範圍的模擬實驗之路徑規劃時間統計圖

2.6 機器手臂之模擬範例

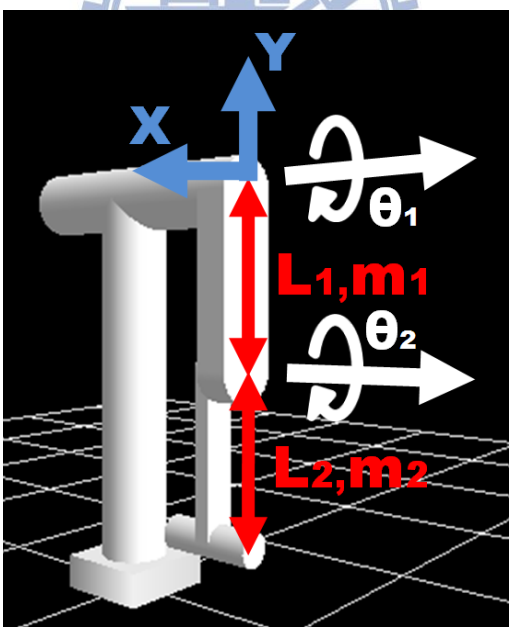
本節將使用 D++演算法應用於機器手臂的軌跡規劃上[47]。在 2001 年時，Wang[48]等人使用了梯度計算方法以解決機器手臂的舉重最佳化問題。他們的研究目標為如何令機器手臂以最省力的方式，以舉起所能負荷的重物。Wang 等人分別展示了成功地令一軸、三軸、與六軸的機器手臂舉重模擬結果。然而梯度法需要事先列出並計算複雜的數學公式，另外也要給定合適的初始值方能收斂至全域最佳解。故這種方式雖然可找到很好的結果，但僅限於一次計算結果後便不斷重複執行同一動作的機器手臂應用，不適合於需自我決策與即時反應的機器人應用上。同樣於 2001 年時, Rosenstein[49]等人提出了直接方針法建立了一組三軸機器手臂舉重學習平台。藉著設計適當的參數式方針與提供合適的手端軌跡中繼點，Rosenstein 等人提出的方法可有效地找出良好的結果。然而需要人工設定的方針與中繼點，以及同樣冗長的試誤學習過程，同樣讓這類方法並不適用於需自我決策與即時反應的機器人應用上。另外在 2007 年時，Cheng[50]等人使用了 Dijkstra 演算法，為一組三軸的舉重機器人建立了手臂移動軌跡規劃方法。然而正如之前提到的，Dijkstra 演算法是一種一次性計算的演算法，所以對於機器手臂軌跡規劃問題，特別是高解析度軌跡規劃，其搜尋空間將十分龐大，因此其搜尋時間將非常久，導致機器手臂無法即時反應的動作。

2.6.1 模擬條件

圖二十九中為一組具兩軸之舉重機器手臂，其相關的尺寸、重量與限制條件如下：

1. 連桿長度： $L_1 = 1$ m, $L_2 = 1$ m
2. 連桿重量： $m_1 = 1$ kg, $m_2 = 0.5$ kg
3. 軸之轉動範圍： $-150° \leqq \theta_1 \leqq 150°$, $0° \leqq \theta_2 \leqq 150°$
4. 軸之轉矩限制： $\tau_1 \leqq 50$ Nm, $\tau_2 \leqq 50$ Nm

本節使用 D++演算法的軌跡規劃目標，為使此舉重手臂以手端移動距離較短方式將重物由機器手臂的手端最低點（$\theta_1 = -90°, \theta_2 = 0°$），舉升到手端的最高點（$\theta_1 = 90°, \theta_2 = 0°$），如圖二十九所示。 因此我們將原先應用在 X、Y 平面座標的 D++演算法，改成應用在 θ_1 、θ_2 平面座標。而此 θ_1 、θ_2 座標平面上的障礙物即為超過上述角度與轉矩限制的座標點。



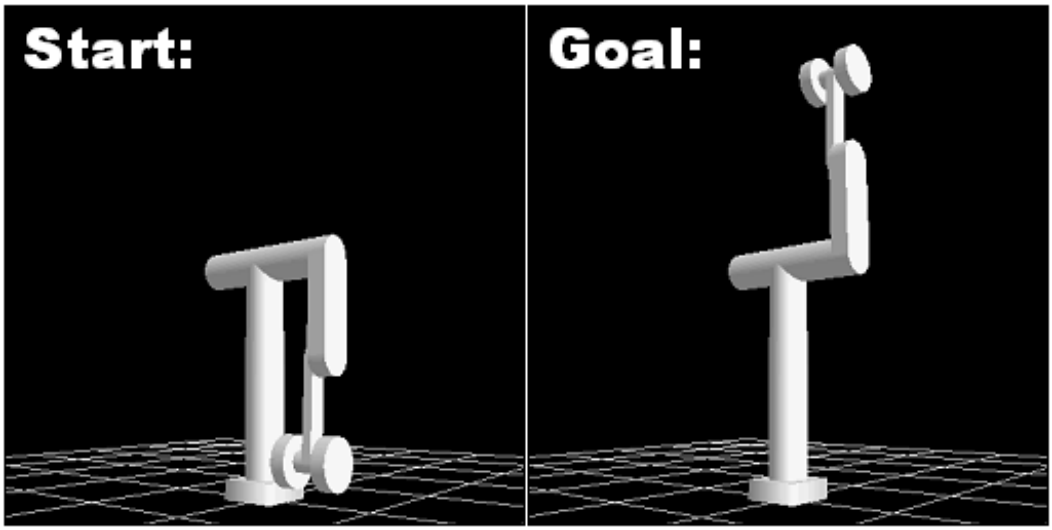
圖二十八、具兩軸之舉重機器手臂

而此舉重機器手臂的手端位置可根據式(2-3)與式(2-4)計算：

$$X_h = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \qquad (2\text{-}3)$$

$$Y_h = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \tag{2-4}$$

式中 X_h 表示機器手臂手端在 X 方向的座標，, Y_h 表示機器手臂手端在 Y 方向的座標。



圖二十九、本章節舉重機器手臂軌跡規劃目標

因此在本問題中， D++演算法的目標函式（cost function）可以式(2-5) 表示：

$$\cos t_C = \cos t_L + \sqrt{(X_C - X_L)^2 + (Y_C - Y_L)^2} \tag{2-5}$$

式中 $cost_C$ 表示 D++演算法中的子節點目標函數值，$cost_L$ 表示上一節點的目標函數值，X_C 表示子節點的手端 x 方向的位置，X_L 表示上一節點的手端 x 方向的位置，Y_C 表示子節點的手端 y 方向的位置，Y_L 表示上一節點的手端 y 方向的位置。為了簡化問題，我們根據了靜力學理論列出了機器手臂每一軸承受力矩之公式，如式(2-6)與 (2-7)所示。

$$\tau_1 = g\left[\left(\frac{m_1}{2} + m_2 + M\right)L_1 \cos\theta_1 + \left(\frac{m_2}{2} + M\right)L_2 \cos(\theta_1 + \theta_2)\right] \tag{2-6}$$

$$\tau_2 = g\left(\frac{m_2}{2} + M\right)L_2 \cos(\theta_1 + \theta_2) \tag{2-7}$$

式中 M 表示負重的重量，g 表示重力加速度常數。接著本問題中的障礙物定義如下：

$$|\tau_1| > 50Nm \tag{2-8}$$

$$|\tau_2| > 50Nm \tag{2-9}$$

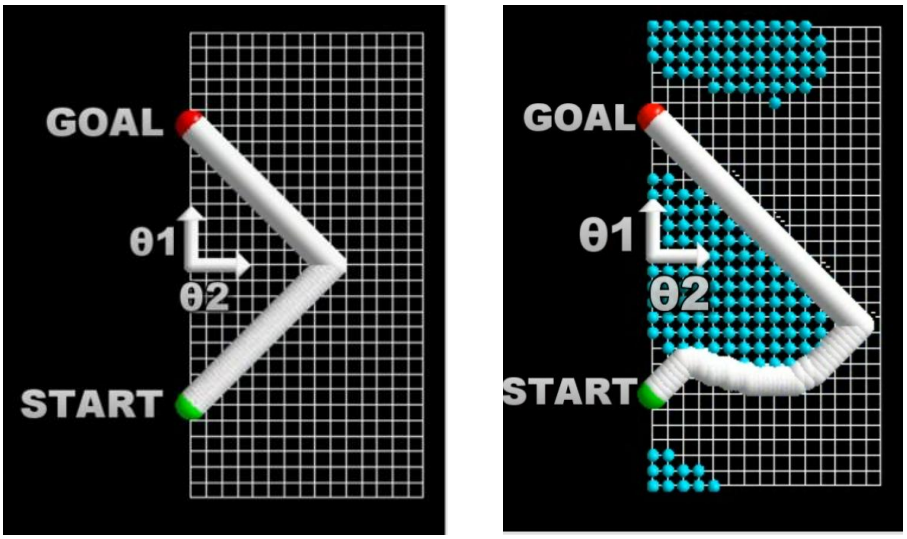
$$\theta_1 > 150^\circ \quad \text{or} \quad \theta_1 < -150^\circ \tag{2-10}$$

$$\theta_2 > 150^\circ \quad \text{or} \quad \theta_2 < 0^\circ \tag{2-11}$$

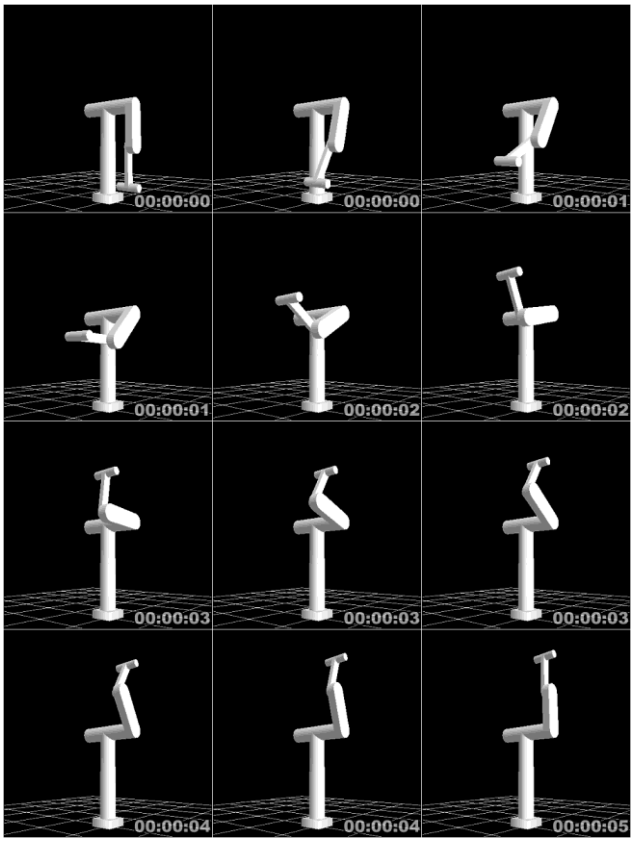
2.6.2 模擬結果

因此根據上述之理論與公式，使用 D++演算法計算機器手臂的舉重軌跡之模擬結果如圖三十所示，其中圖三十(a)為負重 0kg 之模擬結果，而圖三十(b) 為負重 4kg 之模擬結果。這兩個模擬結果均具有 10 毫秒的循環週期，且每軸之解析角度(θ_1、θ_2)為 1 度。而圖三十中，機器手臂的舉起 0kg 與 4kg 負重的模擬情形，則分別如圖三十一與圖三十二所展示。從圖三十的結果比較我們可以看到，圖三十(b)中的路徑比起圖三十(a)的路徑繞行地更遠，這是因為圖三十(b)中的負重較大，相較於圖三十(a)的例子將更容易超過轉矩限制。所以圖三十(b)中 θ_1 與 θ_2 的平面座標上，相較於圖三十(a)的平面座標多了許多的障礙物(圖三十(b)中較小的圓點)。因此圖三十(b)中路徑必須繞過較多的障礙物方能抵達終點。

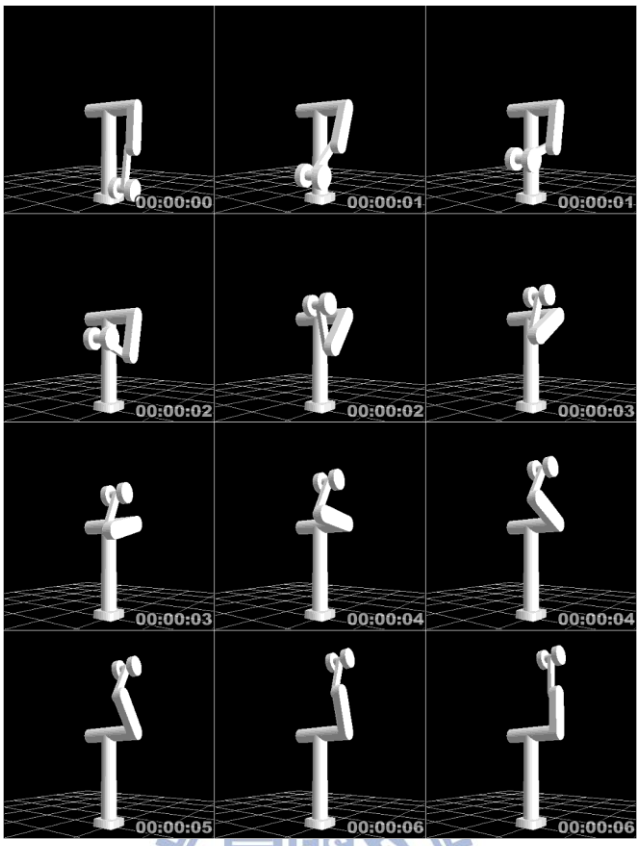
而觀察機器手臂舉起負重的模擬結果，圖三十一的例子由於無負重，所以為了快速從起點到達終點，機器手臂的手端位置離肩部(θ_1 之軸處)較遠，也可以在轉矩限制下完成舉重動作。而圖三十二的例子由於有 4kg 的負重，所以為了在轉矩限制下完成舉重動作，機器手臂的手端位置必須離肩部較近以降低力矩臂與力矩，方可順利地從起點到達終點。



圖三十、於卡氏座標系統的模擬結果比較(a) 負重為 0kg 之結果(b) 負重為 4kg 之結果



圖三十一、機器手臂舉起負重 0kg 的模擬情形(10 毫秒的循環週期，且每軸之解析角度為 1 度)



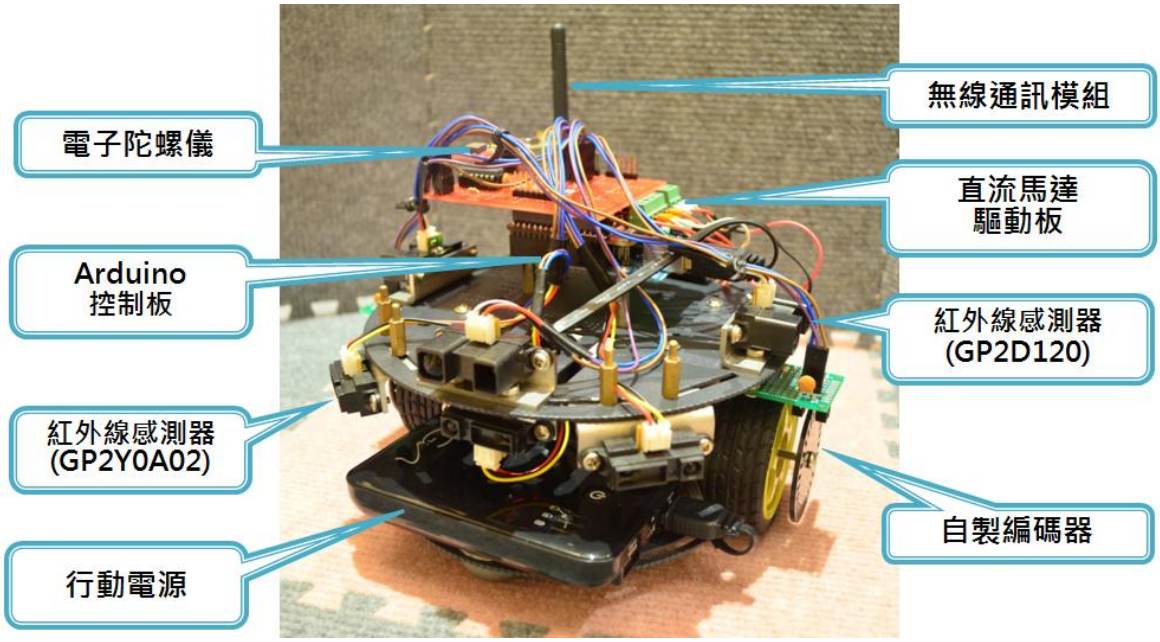
圖三十二、機器手臂舉起負重 4kg 的模擬情形(10 毫秒的循環週期，且每軸之解析角度為 1 度)

透過本模擬實驗的進行，我們成功且有效地運用了 D++演算法於機器手臂的路徑規劃上，並具有快速反應使用者命令需求的能力，避免因需大量計算而導致機器手臂響應過慢，以致無法處理動態環境等問題。對於未來用於家庭、或醫療照顧等需即時反應的智慧型機器手臂應用，可具有較高的實用性。

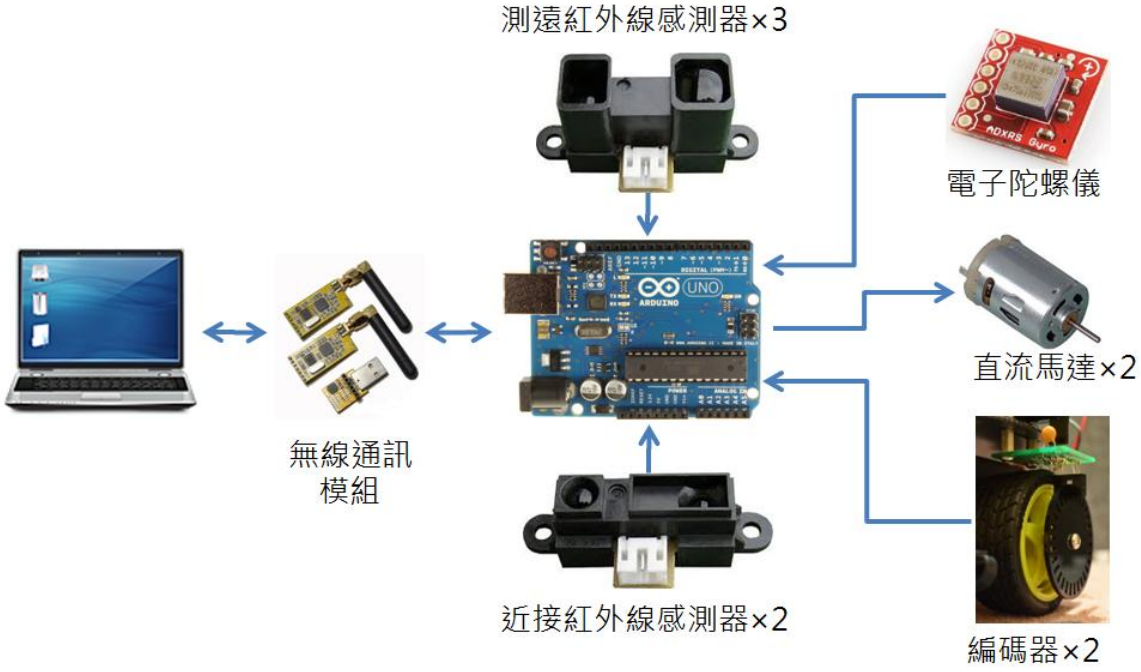
三、D++演算法與移動機器人之整合實作

3.1 系統架構

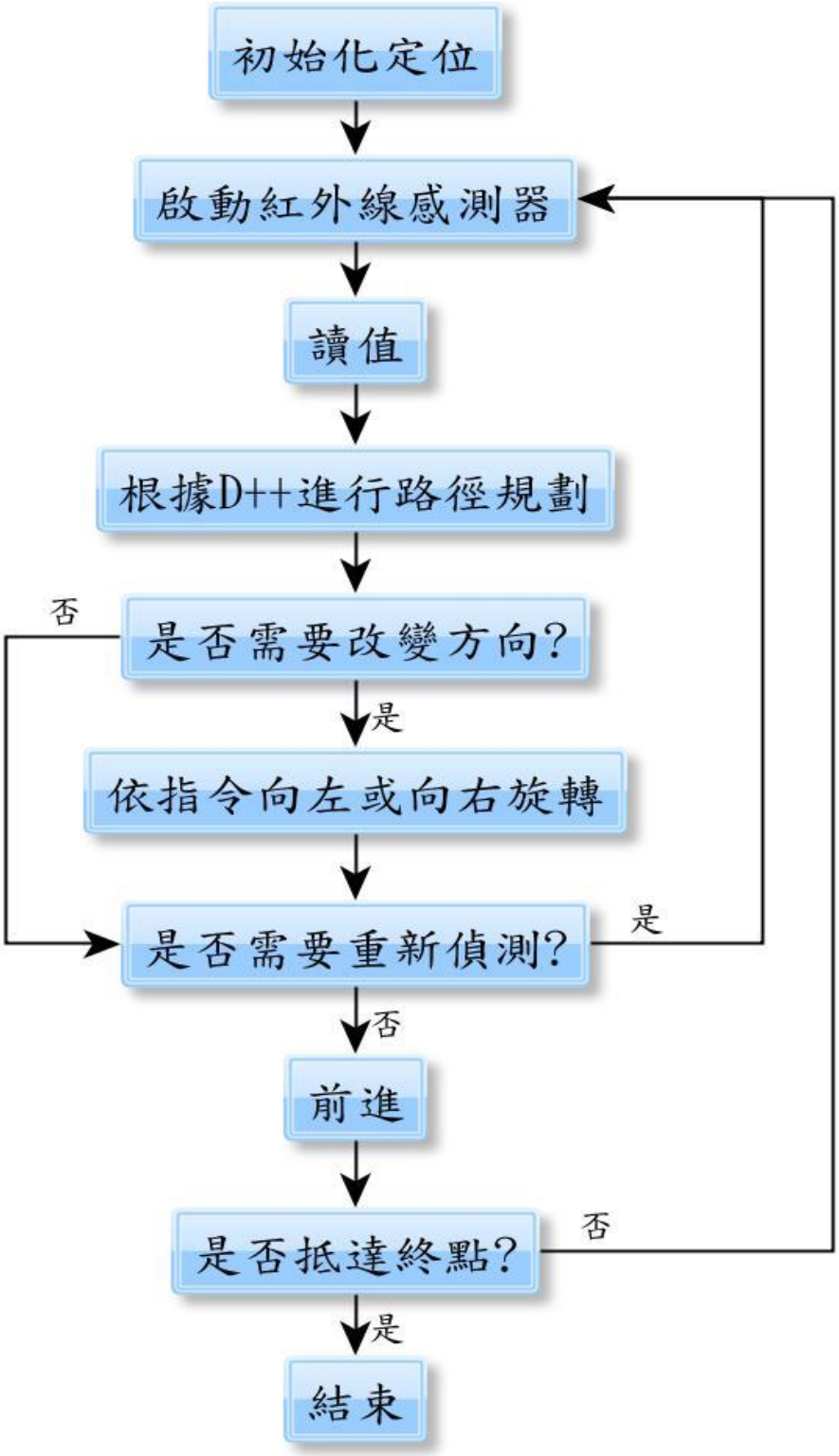
在前兩章節中，我們介紹了 D++演算法的基本概念與流程，並以電腦模擬程式驗證了 D++演算法的可行性。本章將介紹本研究如何實際製作移動機器人，並於一實驗性室內環境進行路境規劃與移動，以證明 D++演算法具有很高的實用性。本研究將設計製造具兩輪之移動機器人(如圖三十三所示)。該移動機器人使用兩個直流馬達驅動兩個輪子轉動，並使用 Arduino 開發板的 PWM 輸出功能控制直流馬達的轉速與與轉向。為了精簡移動機器人的製作成本與構造，D++演算法等高階控制流程將透過個人電腦(PC)或筆記型電腦(NoteBook)實現；而電腦與 Arduino 開發板之溝通則以 RF 無線電傳輸實現。其系統架構如圖三十四所示。Arduino 開發板負責接收 RF 無線電傳輸指令，並擷取紅外線感測器的訊號回報給遠端控制電腦規劃路徑，以獲得各種決策如前進或轉向等。



圖三十三、具兩個驅動輪(左右)與一個萬向輪(前)之移動機器人



圖三十四、移動機器人之系統架構圖

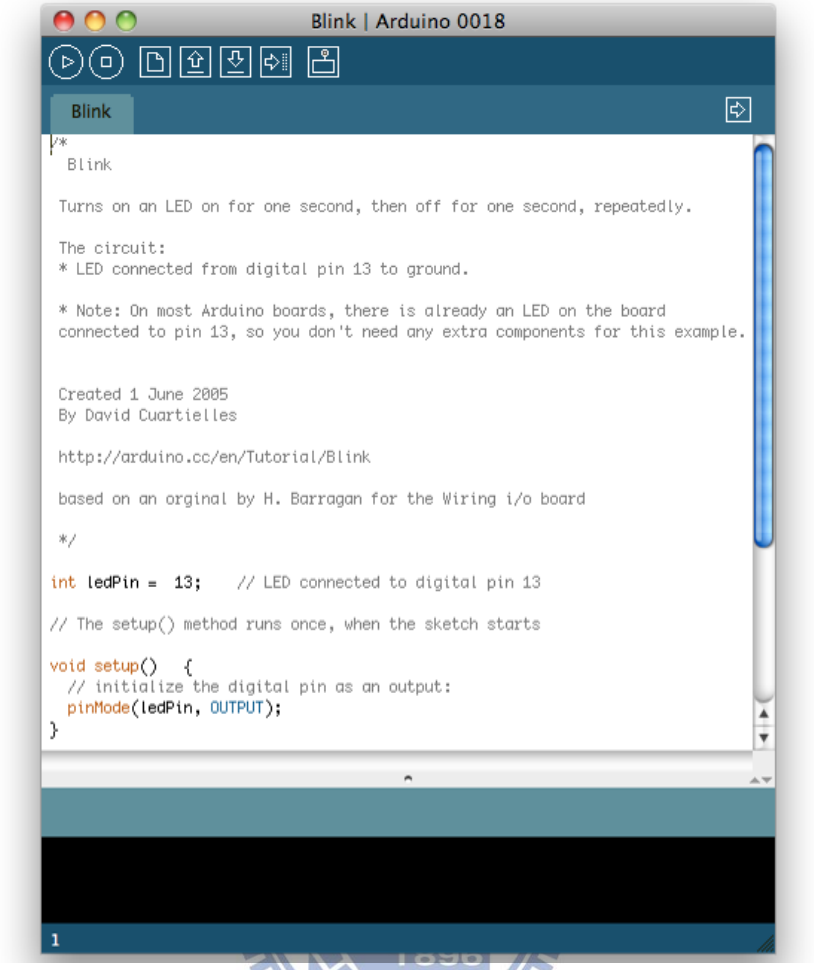


圖三十五、機器人控制流程

3.2 Arduino 控制開發板

Arduino[51]是一個的「開放式硬體（open source）」微電腦控制板，在 2005 年 1 月由米蘭互動設計學院的教授 David Cuartielles 和 Massimo Banzi 所設計出來了，原始構想是希望讓設計師及藝術家們，透過 Arduino 很快的學習電子和感測器的基本知識，快速的設計、製作作品的原型，很容易與目前設計系所學的 FLASH, MAX/MSP, Virtool 等軟體整合，使得虛擬與現實的互動更加容易。互動的內容設計才是設計師的主要訴求，至於怎麼拼湊一個單晶片開發板，或是當中涉及如何構築電路之類的知識，就並非設計師需要了解的，因此非常適合不具電子背景的人使用，以設計出各種不同的互動裝置。

Arduino 控制板主要是採用 Atmel 公司的 AVR 系列微處理器。它的硬體很簡潔，沒有什麼特殊的設計，除了開放式硬體架構之外，重點在於它提供了一個基於 C/C++的程式語言來控制 Arduino，並且有 Windows, Mac OS X 和 Linux 等跨平台環境的程式編輯工具(如圖三十六)。除了當成可獨立運作的微電腦控制器之外，它也能透過許多愛好者開發出的程式模組，和電腦上的其他程式語言通訊，例如：Flash ActionScript, Processing, Python, PHP,…等等。微電腦自動控制的專業人士，大多優先選擇「組合語言」來撰寫程式，因為組合語言的執行速度快，而且佔用的記憶體空間小。如果嫌高階語言不專業，使用者也可以安裝 Atmel 公司的 AVR Studio 開發工具，用組合語言或者 AVR C 語言撰寫 Arduino 的程式。



圖三十六、Arduino 軟體開發除錯環境(來源：[51])

因為本研究所建置的移動機器人，還需要使用到專為 Arduino 開發的馬達驅動板（如圖四十六所示），該馬達模組可直接層疊至 Arduino Uno 上(如圖三十三所示)，不需另外的傳輸線，故我們採用的是 Arduino Uno 這塊開發控制板。Arduino Uno 為使用 ATmega328 微控器的控制板。它具有 13 個數位輸入/輸出埠，其中 Pin3、5、6、9、10、11 具有 PWM 輸出功能，Pin2,3 具有中斷輸入功能，Pin0, Pin1 為一組串列通訊埠(0=Rx, 1=Tx)；Pin13 具有 LED 輸出功能。Arduino Uno 還有 6 個類比輸入埠(A0~A5)。Arduino Uno 使用 16MHz 的振盪器，並具有 128K 的記憶體。至於下載程式的介面，則是使用 ATmega8U2 晶片來處理 USB 通訊

(B-Type 接頭)，並可支援多平台作業系統 windows/Linux/Mac。電源方面使用 2.1mm 的 Power Jack 接頭，可輸入 6~20V 的 DC 電源，官方則建議使用 7~12V 的 DC 電源。然而為了電源的長續航力，本移動機器人平台使用 5V/1A、具 USB 埠的行動電源來提供 Arduino 控制板所需要的電源。

圖三十七、Arduino UNO 控制板(來源：[51])

在本研究中，我們使用 Arduino Uno 來擔任移動機器人本體的硬體控制器。接著我們針對本研究所使用到 Arduino 控制板的主要功能，做出更進一步的介紹。在單晶片控制技術中，輸入輸出是單晶片最基本的介面，通過這個介面，單晶片可以用數位或類比的方式，和周邊電路連接，並進行控制電路訊號和訊號檢測。在上一段中，我們已概略介紹 Arduino 開發板具有哪些輸入輸出功能，接著我們針對這些功能，來說明在本研究中所應用的方式：

(1) 數位輸出輸入：

在本研究中，我們使用數位輸出輸入，來取得編碼器的輸入訊號，並控制馬達驅動器的輸出訊號。

(2) 類比輸入：

在本研究中，測量障礙物的位置主要是由紅外線感測器所偵測，而我們使用的紅外線感測器之感測距離，為使用類比電壓輸出。故我們使用 Arduino 開發板上的 A0~A5 來接收紅外線感測器與電子陀螺儀的訊號。在 Arduino 開發板上，類比訊號為 10bit 的資料，故可將得到的類比電壓值細分為 0~1023 數位資料。

(3) PWM：

在本研究中，PWM 主要是控制移動機器人的 DC 馬達轉速，每一顆 DC 馬達需要 1 組 PWM 控制其轉速，1 條數位輸出控制其正反轉。所以在本研究中，我們需要 2 組 PWM 來控制兩個 DC 馬達的轉速，2 Pin 的數位輸出埠控制兩個 DC 馬達的正反轉，。

(4) 計時中斷：

在本研究中，計時中斷主要負責機器人的馬達轉動的同步轉動控制、移動距離控制與轉向時的角度控制。為本研究移動機器人運動控制最重要的部分。

(5) 輸入觸發中斷：

在本研究中，我們使用了 2 pin 的輸入觸發中斷，來取得馬達編碼器傳至 Arudino 板的訊號，以獲得目前機器人的移動距離，進而換算機器人目前之位置。使用觸發中斷埠的優點是能保證不漏掉訊號。

(6) 串列通訊埠：

在本研究中，串列通訊埠主要是透過無線電 RF 傳輸模組，與遠端控制電腦溝通。因此 Arduino 控制板負責移動機器人馬達驅動與感測器接收等硬體工作，而電腦則負責較複雜的路徑規劃計算工作，以達到多工目的。

3.3 運動控制用之硬體

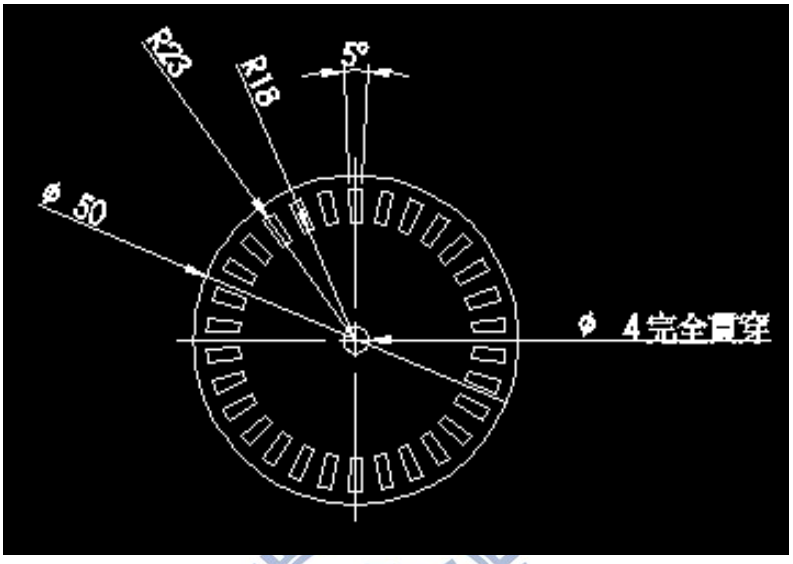
3.3.1 自製編碼器

本節說明本研究所使用之自製光學編碼器的工作原理及應用，它被安裝在輪型機器人左右兩側車輪軸心上，當車輪轉動時將產生穩定的脈衝輸出，在單位時間內計數脈衝個數可以轉換得知左、右車輪移動的距離，進而得知機器人的方位及座標，並導引機器人到達期望的目標，這種方法被稱為車輪測程法。

光學編碼器是由一組光遮斷器及齒輪狀之圓盤所組合而成(如圖三十八)，當固定在車輪上的圓盤轉動時，光遮斷器之光通過圓盤齒隙沒有被阻擋的話，其輸出為高電位，藉由提升電阻拉為高電位。反之則輸出為低電位。

圖三十八、本研究所使用之自製光學編碼器

本研究所使用的光學編碼器，圓盤上平均分配 30 個齒，每旋轉一周可產生 30 個脈波或 60 個邊緣觸發，參考圖三十九所示。



圖三十九、本研究之自製編碼盤尺寸圖

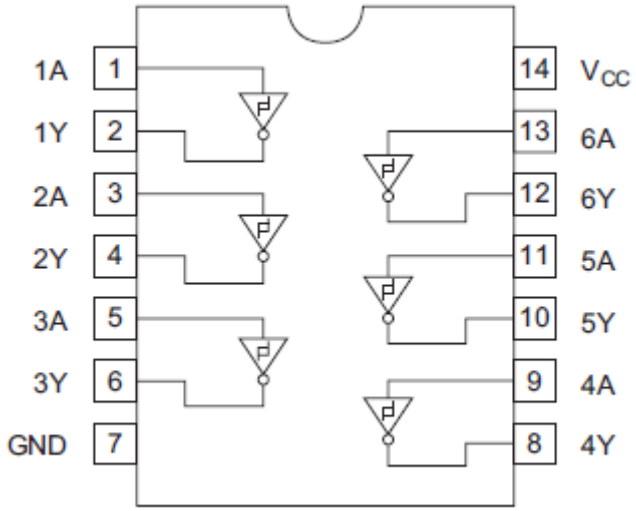
若左右兩車輪的輪距為 $2b$，車輪半徑為 r，車輪每旋轉一周，編碼器送出 N 個脈波，假設兩車輪同方向同步轉動了 n 個脈波，輪型機器人在直線上移動的距離可計算如下:

$$d = \frac{2\pi nr}{N} \tag{3.1}$$

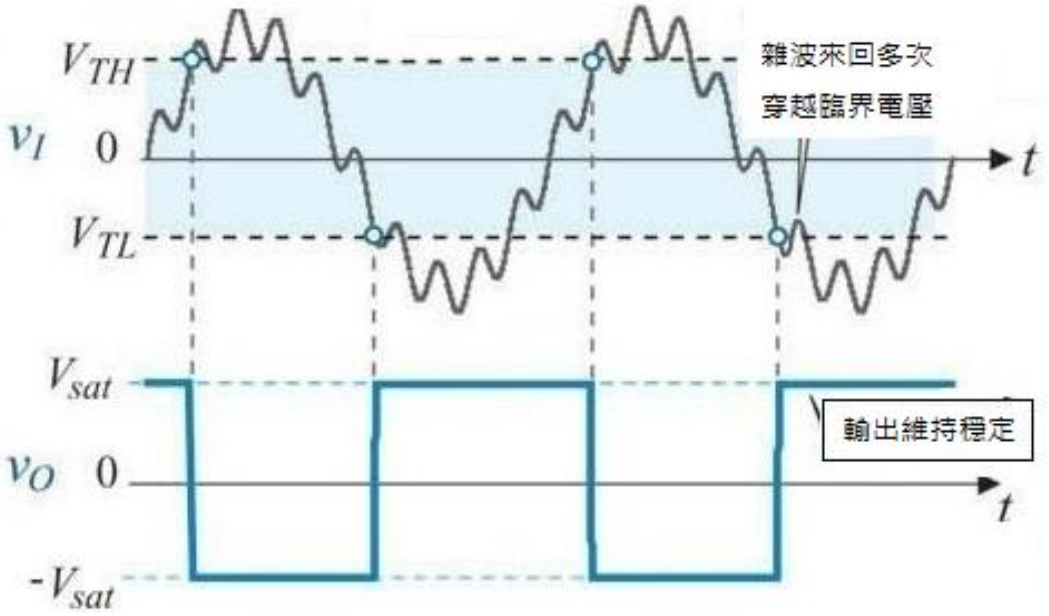
以本研究所採用的輪型機器人及光學編碼器為例，車輪半徑為 30mm，每轉一圈送出 60 個邊緣觸發，在輪胎不打滑的前提下，輪型機器人約向前移動了 188.5 mm 的距離，每個邊緣處發信號的精度為 3.14mm。

由於從光遮斷器所獲取的訊號為弦波而不是方波，加上馬達轉動時容易對週遭電路產生干擾，故若是我們直接擷取從光遮斷器來的訊號，非常容易產生誤判的情形，造成編碼器嚴重的計數誤差。為了得到正確

的編碼器計數值，我們使用史密特觸發反相器 IC－HD74LS14（如圖四十），將從光遮斷器擷取來的不穩定弦波訊號（輸入至 A），透過反相器來整形成穩定的方波訊號（由 Y 輸出），以獲得正確的編碼器計數值（如圖四十一所示）。



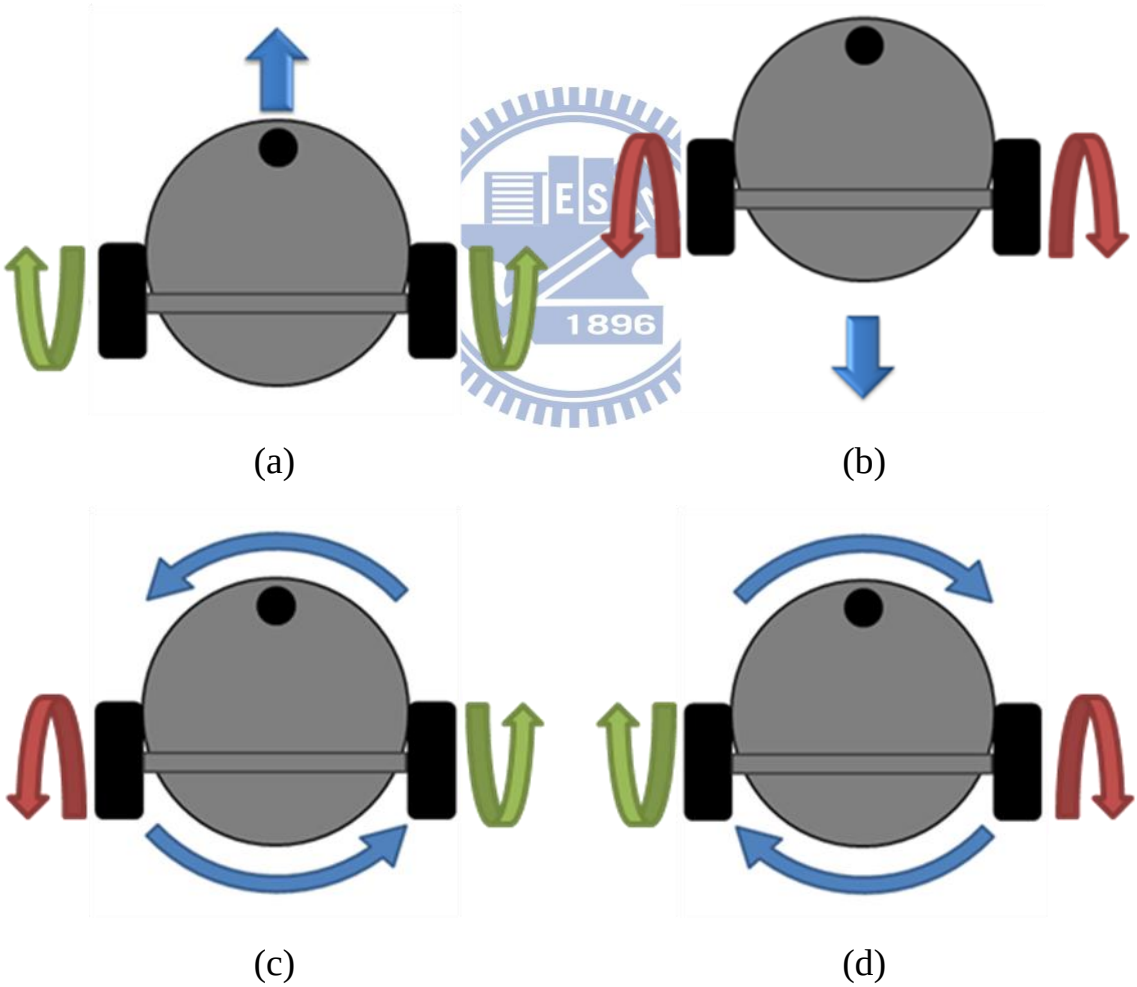
圖四十、HD74LS14 之內部結構圖（來源：[54]）



圖四十一、使用史密特觸發反相器穩定光遮斷器之訊號（來源：[55]）

3.3.2 雙輪同動控制

如同本章第一節所提到，本研究所建立的移動機器人平台，具一個萬向輪(前)與兩個驅動輪(左右)。萬向輪的作用僅為支撐機器人不翻倒，真正令移動機器人的前進、後退、轉向為位於左右的驅動輪。左右驅動輪同時向移動機器人前方轉動時，則移動機器人便往前進；反之左右驅動輪同時向移動機器人後方轉動時，則移動機器人便會後退。若是左右驅動輪一個向前、一個向後轉動，則移動機器人便會根據驅動輪轉動的方向，而向左或向右轉向（如圖四十二所示）。



(a) (b)

(c) (d)

圖四十二、本研究移動機器人之運動控制邏輯(a)前進(b)後退(c)左轉(d)右轉

然而以上的控制邏輯，必須在左右兩個驅動輪具有相同轉動位移或速度的條件下，方能達成。若是左右兩個驅動輪不同動，便會造成斜向地前進或後退，或是非以移動機器人為中心轉向。由於本研究之移動機器人平台，並未架設外部的機器人絕對位置定位裝置（如 GPS 系統、CCD 照相機或室內網路定位系統）；因此本研究之移動機器人的定位方式，為透過編碼器與上述前進、後退、轉向的動作，自行計算其相對的移動位置，以換算可能的絕對位置。因此若是移動機器人因為左右兩個驅動輪不同動，造成斜向地前進或後退，或是非以其中心轉向，便會使得換算的位置與實際有很大的誤差。因此我們利用 PID 控制法則，透過編碼器的訊號回授，來控制兩個驅動輪以相同轉動位移與速度同時轉動，以達到我們利用編碼器與相對位移換算其絕對位置的目的。

PID 控制法則[60][61]問世至今已有近 60 年的歷史，此種控制具有結構簡單、穩定性好、工作可靠、調整方便等四項優點而成為工業控制技術的工具。假若受控制對象的組態與參數無法完全掌握，或是無法獲知精確的數學模型時，系統的控制器的參數就必須依靠經驗和實際調試來確定，此時，應用 PID 控制技術最為方便。PID 控制法則由比例增益(P)、積分增益(I) 和微分增益(D)組成，通過 Kp， Ki 和 Kd 三個參數的設定，使得在單一輸入及單一輸出的控制下可以達到非常好的穩定性。我們可以認為積分是過去偏差的累積值，比例是現在偏差的大小，微分則是針對將來的偏差變化動向做各種對應。因此控制系統的設計重點是如何決定 Kp、Ki、Kd 這三個參數值，我們將之稱為 PID 控制器的調整，就是替控制器參數尋找一個最佳設定值。而比例增益(P)、積分增益(I) 和微分增益(D)的功用說明如下：

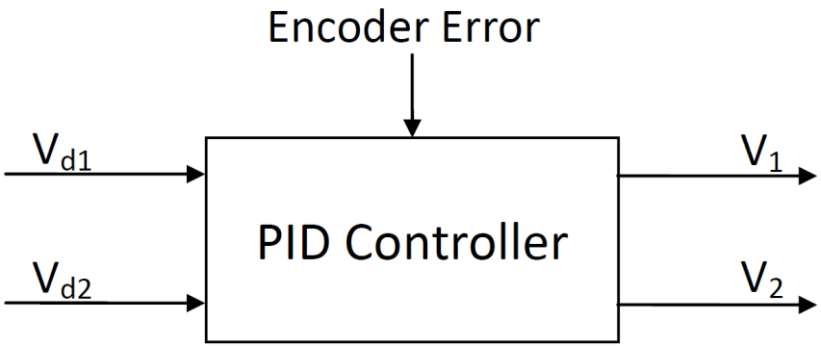
(1) 比例（P）控制：是一種最簡單的控制方式。其控制器的輸出與輸入誤差信號成比例關系。當僅有比例控制時系統輸出存在穩態

誤差（Steady-state error）。

(2) 積分（I）控制：在積分控制中，控制器的輸出與輸入誤差信號的積分成正比關系。積分項對誤差取決於時間的積分，隨著時間的增加，積分項會增大。這樣，即便誤差很小，積分項也會隨著時間的增加而加大，它推動控制器的輸出增大使穩態誤差進一步減小，直到等於零。因此，比例+積分(PI)控制器，可以使系統在進入穩態後無穩態誤差。

(3) 微分（D）控制：在微分控制中，控制器的輸出與輸入誤差信號的微分（即誤差的變化率）成正比關系。比例項的作用僅是放大誤差的幅值，而微分項能預測誤差變化的趨勢，這樣，具有比例+微分的控制器，就能夠提前使抑制誤差的控制作用等於零，甚至為負值，從而避免了被控量的嚴重超調。所以對有較大慣性或滯後的被控對象，比例+微分(PD)控制器能改善系統在調節過程中的動態特性。

在本研究的雙輪同動 PID 控制問題中，由於是我們使用 Arduino 板上的 PWM 輸出功能來控制直流馬達的轉速，所以同動 PID 控制中的輸入為預期左右驅動輪的馬達轉速，輸出為使用 PID 調整後的馬達轉速，而回授訊號為左右驅動輪上的編碼器數值（如圖四十三所示）。



圖四十三、本研究雙輪同動 PID 控制方塊圖

則我們可使用 PID 控制理論，以進行左右驅動輪的馬達同動的閉迴路伺服控制，如式(3-2)與(3-3)所示：

$$V_L = V_{command} - Kp_L \times pError - Ki_L \times iError - Kd_L \times dError \quad (3\text{-}2)$$

$$V_R = V_{command} + Kp_R \times pError + Ki_R \times iError + Kd_R \times dError \quad (3\text{-}3)$$

在此 V_L 為左輪馬達的新轉速輸出，V_R 為右輪馬達的新轉速輸出，$V_{command}$ 為預期兩顆馬達的轉速，*Kp* 為比例控制的參數值，*Ki* 為積分控制的參數值，*Kd* 為微分控制的參數值；*pError*、*iError*、*dError* 分別為比例、積分、微分誤差。這三組誤差回授值，我們根據增量式 PID 理論來計算，其計算公式分別如式(3-4)~(3-6)所式：

$$pError = Encoder\_L - Encoder\_R \quad (3\text{-}4)$$

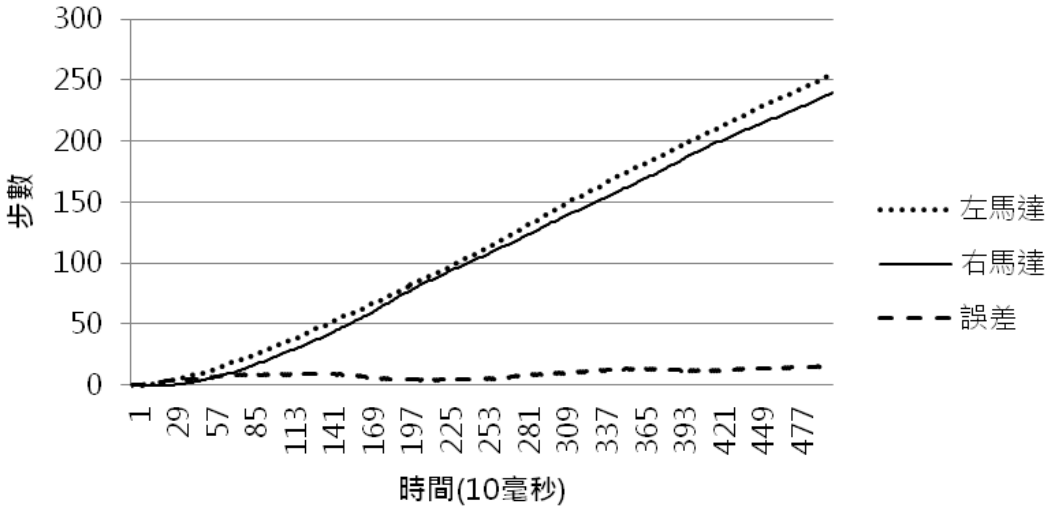
$$iError = iError + pError \quad (3\text{-}5)$$

$$dError = pError - Last\_pError \quad (3\text{-}6)$$

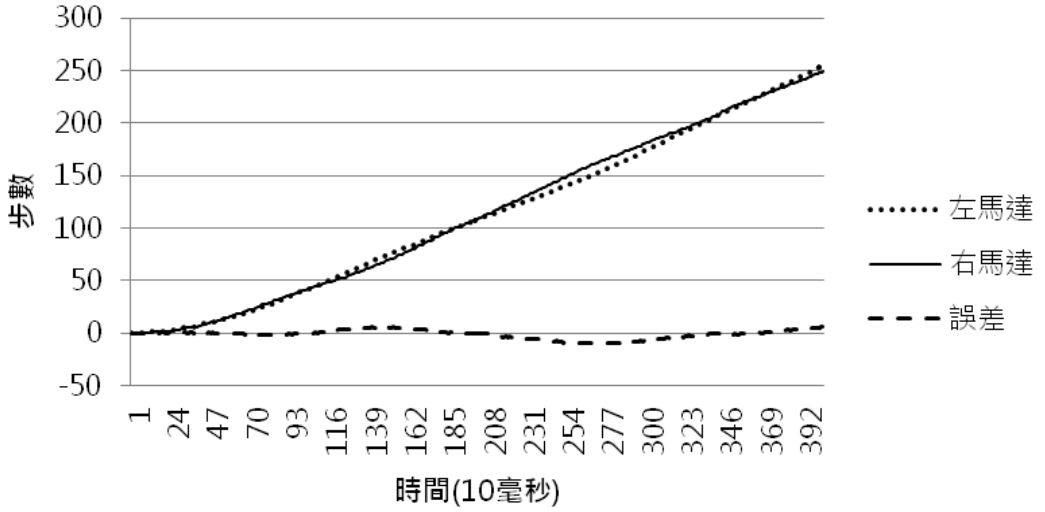
式中 *Encoder_L* 為左輪馬達的編碼器值，*Encoder_R* 為右輪馬達的編碼器值，而 *Last_pError 為上一週期點的 pError 值*。因此如式(3-2)與(3-3)所示，當左輪馬達超前右輪馬達時，*pError 為正值*，故左輪馬達便會比預期轉速 $V_{command}$ 小，而右輪馬達會比預期轉速 $V_{command}$ 大，所以右輪馬達轉速會比左輪馬達略大而逐漸追上其轉動位移。反之當左輪馬達落後右輪馬達時，*pError 為負值*，故左輪馬達便會比預期轉速 $V_{command}$ 大，而右輪馬達會比預期轉速 $V_{command}$ 小，所以左輪馬達轉速會比右輪馬達略大而逐漸追上其轉動位移。藉此達到左右驅動輪同動的目的。

接著我們實際將 PID 同步控制理論撰寫到移動機器人的 Arduino 開發板中，並命令左右馬達同時轉動 250 個編碼器脈波步數，將各自轉動過程的編碼器回授數值紀錄如圖四十四、圖四十五所示。由圖四十四中的結果可以看到，因為未對兩顆馬達做同步控制，所以轉動到最後兩顆

馬達的轉動誤差已到達 20 的編碼器脈波數。而使用了實現了本節提出的 PID 同步控制理論後，最後兩顆馬達的轉動誤差僅僅不到 5 個脈波數。此實驗結果證明本節提出的 PID 同步控制理論，確實讓移動機器人的兩顆馬達完成同步轉動的目標。



圖四十四、無使用 PID 同步控制之馬達運轉結果



圖四十五、使用 PID 同步控制之馬達運轉結果

3.3.3 直流馬達驅動板

在本研究中，我們使用了內嵌 L298P（大功率馬達專用驅動晶片）的直流馬達驅動板(如圖四十六所示)，來控制移動機器人兩個後輪進行不同轉速的正轉與反轉，以達到上一章所述之移動機器人前進、後退、轉向的動作。

圖四十六、具 L298P 之直流馬達驅動板(來源：[56])

L298P 為將 2 個獨立的橋氏驅動電路裝在一個 IC 內，可控制兩顆直流電刷馬達，圖四十七為 L298P 腳位圖，而

表 4 說明其腳位名稱與功能。L298P 的主要特性如下：

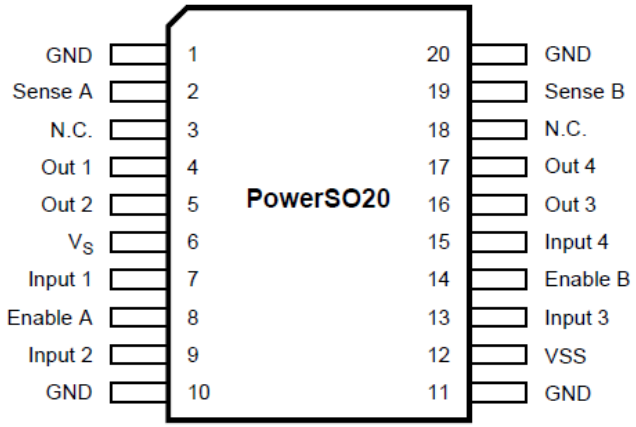
(1) 在 IC 輸入腳位，加入 0 或 1 的邏輯位準信號，以此作輸入信號的組合來作不同的控制，例如正轉、反轉、停止、快速制動等四種功能。

(2) 0 的邏輯位準信號可達 1.5V，可有效地降低信號干擾的影響。

(3) IC 工作電壓與馬達驅動電壓可獨立輸入，可作伺服控制；IC 工

作電壓範圍為 4.5~7V，馬達驅動電壓範圍則為 0~46V。

(4) 最大連續輸出電流為 2A，最大瞬間電流為 3A。



圖四十七、L298P 外觀與尺寸（來源：[57]）

表 4、L298P 接腳功能表

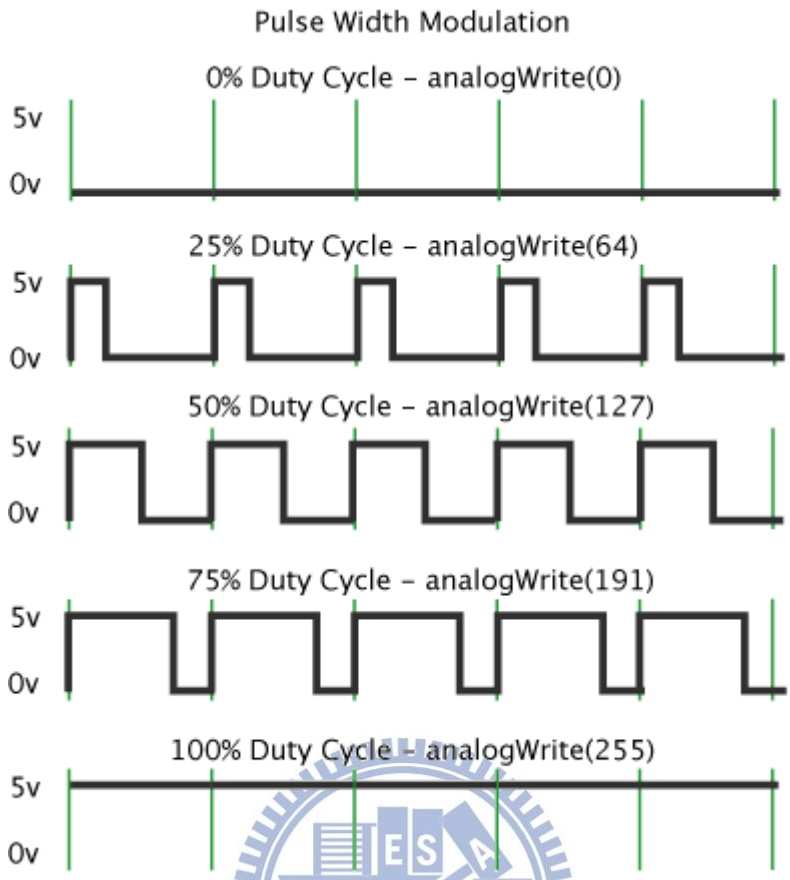
腳位號碼	符號	功能說明
1	GND	接地
2	SENSE A	電流控制腳位
3	N.C.	空腳位
4	OUTPUT 1	第一輸出端
5	OUTPUT 2	第二輸出端
6	V_S	馬達電壓輸入(0~46V)
7	INPUT 1	第一輸入端
8	ENABLE A	致能 OUTPUT 1 與 OUTPUT 2
9	INPUT 2	第二輸入端
10	GND	接地
11	GND	接地
12	Vss	邏輯電壓輸入(4.5~7V)
13	INPUT 3	第三輸入端
14	ENABLE B	致能 OUTPUT 3 與 OUTPUT 4
15	INPUT 4	第四輸入端
16	OUTPUT 3	第三輸出端
17	OUTPUT 4	第四輸出端
18	N.C.	空腳位
19	SENSE B	電流控制腳位
20	GND	接地

L298P 是由 Pin7、Pin9(A)與 Pin13、Pin15(B)中輸入邏輯位準訊號，使得連接馬達兩端的 Pin4、Pin5(A)與 Pin16、Pin17(B)產生不同流向的電壓，使得兩顆馬達進行正、反轉等動作。

表 5 說明輸入邏輯位準訊號與輸出電壓的四種不同排列組合的真值表，以及馬達所對應的動作。而要控制直流馬達以不同轉速運轉，只需使用 Arduino 板上的 PWM 功能，輸入不同頻寬的 5V 電壓於 Input1 與 Input2 之腳位，便可產生等效於不同的輸入電壓於直流馬達，而產生不同的轉速(如圖四十八所示)。

表 5、L298P 真值表

ENABLE	INPUT1	INPUT2	OUTPUT1	OUTPUT2	馬達動作
H	H/L	H/L	H	H	快速制動 (Brake)
H	L	H	L	H	正/反轉 (CW/CCW)
H	H	L	H	L	反/正轉 (CCW/CW)
L	X	X	L	L	自由移動

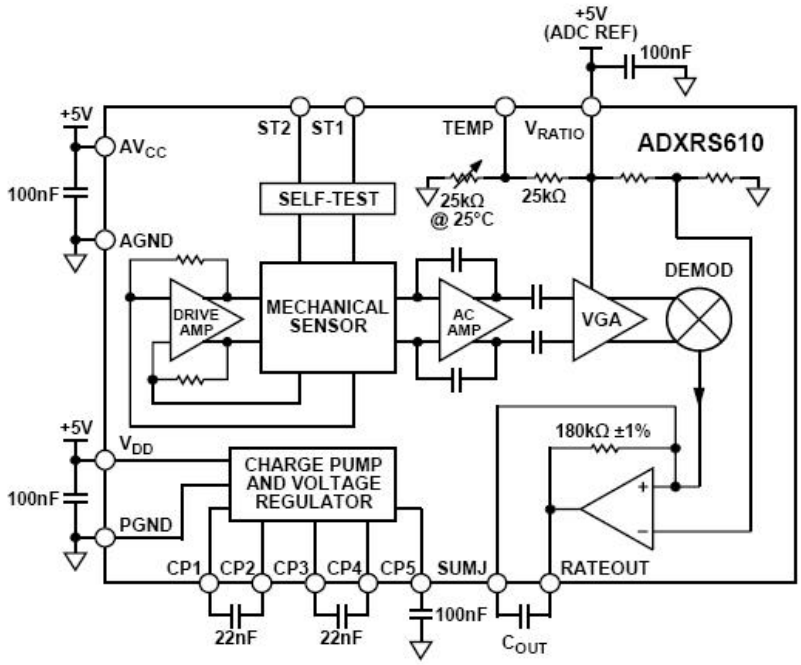


圖四十八、不同 PWM 輸入產生的等效電壓輸入（來源：[51]）

3.3.4 電子陀螺儀

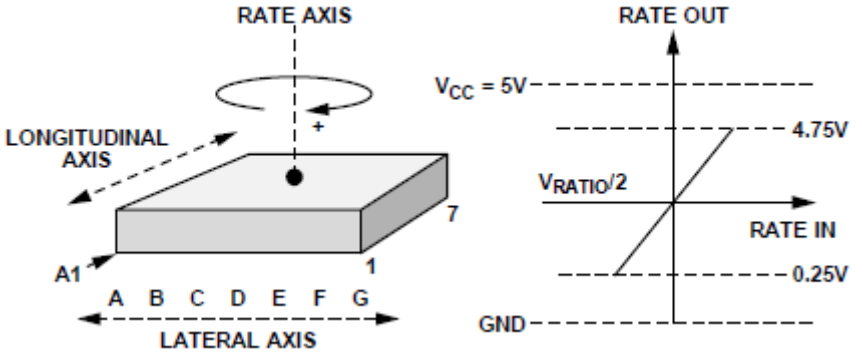
移動機器人在旋轉的運動過程中，有可能會因為重心轉移或是因為灰塵而造成輪胎打滑，導致機器人產生姿態上的誤差，故本研究選用Analog Devices公司的陀螺儀ADXRS613（如圖四十九）。利用陀螺儀ADXRS613計算機器人旋轉時的角度。圖五十為陀螺儀ADXRS613的內部方塊圖。

圖四十九、陀螺儀 ADXRS613 之外觀圖（來源：[58]）



圖五十、陀螺儀 ADXRS613 之內部方塊圖（來源：[58]）

陀螺儀ADXRS613量測的角速度的參考軸如圖3.39所示，垂直IC包裝的文字面；那麼陀螺儀順時針繞著參考軸的角速度為正值，接腳RATEOUT輸出電壓值與角速度成正比。



圖五十一、ADXRS613 所量測角速度的方向與輸出電壓大小參考圖（來源：[58]）

表 6 是陀螺儀 ADXRS613 的規格特性：典型的零點電壓(靜止不動)是以 2.5V 為參考電壓；典型的輸出電壓與角速度的關係是 12.5mV/(°/s)；當輸出電壓大於 2.5V 時，角速度為正值，反之，角速度為負值；假設角速度為 150°/s，那麼輸出電壓會是

$$2.5+150\times0.0125=4.375(V) \tag{3.7}$$

瞭解了以上的關係後，將量測到的輸出電壓值ω_{ad}轉換成角速度ω的大小：

$$\frac{\omega_{ad}-2.5}{4.375-2.5}=\frac{\omega}{150}\Rightarrow\omega=(\omega_{ad}-2.5)\times150/1.875 \tag{3.8}$$

本研究利用Arduino控制板的10位元類比數位轉換器，將0V-5V陀螺儀的輸出電壓範圍轉成0-1023的數值範圍時，假設零點電壓2.5V的數值是511，那麼角速度為150°/s時的數值會是4.375V/5V×1023≒895。我們可以將上式改成：

$$\frac{\omega_{ad}-511}{895-511}=\frac{\omega}{150}\Rightarrow\omega=(\omega_{ad}-511)\times150/384 \tag{3.9}$$

表 6、陀螺儀 ADXRS613 的規格特性（來源：[58]）

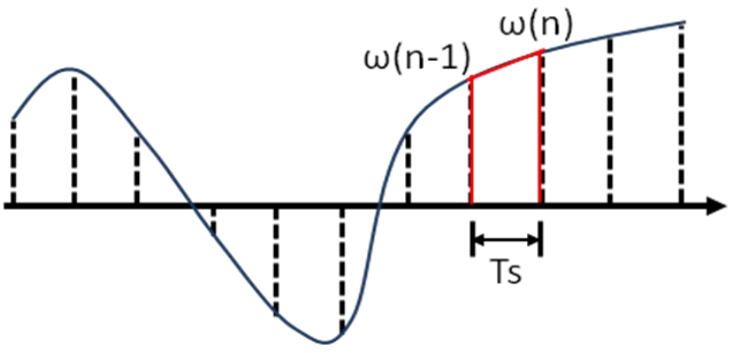
Parameter	Conditions	ADXRS613BBGZ Min	Typ	Max	Unit
SENSITIVITY (RATIOMETRIC)[1]	Clockwise rotation is positive output				
Measurement Range[2]	Full-scale range over specifications range	±150			°/sec
Initial and Over Temperature		11.25	12.5	13.75	mV/°/sec
Temperature Drift[3]			±3		%
Nonlinearity	Best fit straight line		0.1		% of FS
NULL (RATIOMETRIC)[1]					
Null	−40°C to +105°C		2.5		V
Null Drift Over Temperature	−40°C to +105°C			±250	mV
Linear Acceleration Effect	Any axis		0.1		°/sec/*g*

當我們利用(3.9)式知道機器人旋轉的角速度時，計算機器人旋轉角度Δθ可以由角速度的積分計算出來，因此要計算一定的時間內，機器人旋轉角度的變化量Δθ，可以使用梯形軟體積分法（如圖五十二所示）計算出來：

$$\Delta\theta = \int_{t_0}^{t_0+T_s} \omega dt \Rightarrow \Delta\theta(n) = \Delta\theta(n-1) + \left[\omega(n) + \omega(n-1)\right]T_s \tag{3.10}$$

其中，*Ts*代表取樣時間。以上關於式(3.10)的梯形軟體積分法中，只是計算旋轉角度的變化量Δθ而已，如果要知道正確機器人朝向的角度大小，那就還需要給予機器人初始旋轉角度θ_0，則機器人朝向的角度θ(*n*)為：

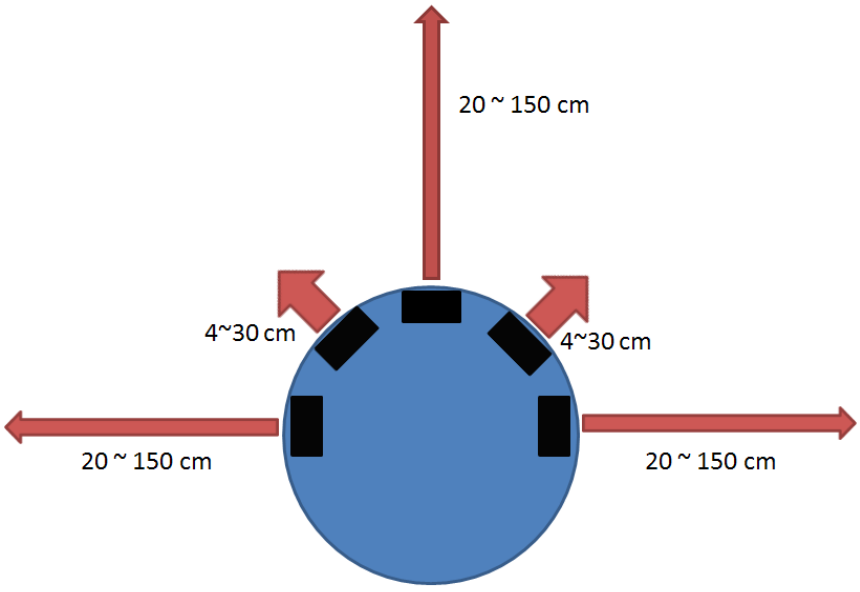
$$\theta(n) = \theta_0 + \Delta\theta(n) \tag{3.11}$$



圖五十二、梯形軟體積分法示意圖

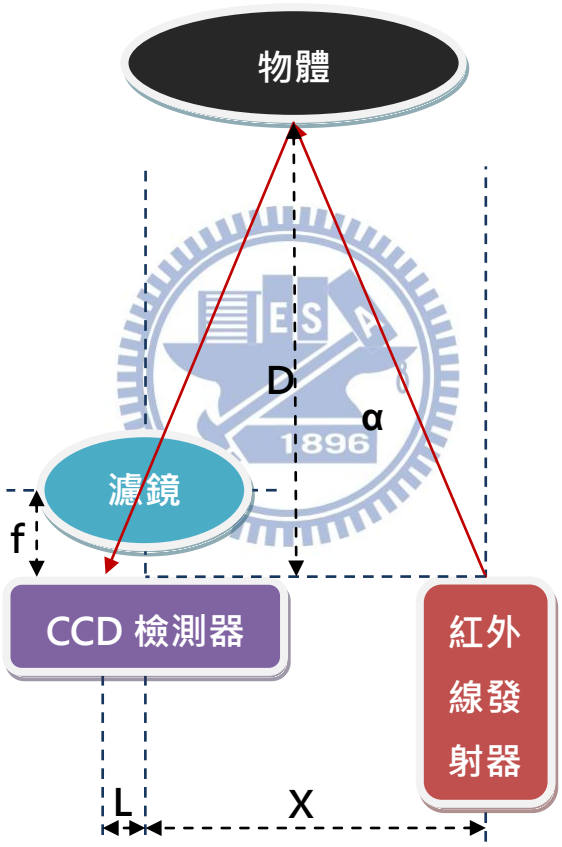
3.4 紅外線感測器

雖然超音波感測器可以有效大範圍偵測障礙物離機器人的距離，但受限於超音波的特性，多個超音波感測器同時量測距離，容易產生干擾問題；另外超音波感測器對於近端障礙物的量測，其準確性也較差。為了讓機器人在行進間，能即時同步量測近端的障礙物狀況，故本研究使用 5 個 Sharp 公司生產的紅外線感測器－GP2D120 與 GP2Y0A02。其中我們使用了 3 個 GP2Y0A02 紅外線感測器，分別安裝在機器人前端左邊 45 度、中間、右邊 45 度之位置(如圖五十三所示)。GP2Y0A02 的偵測範圍為 20cm ~ 150cm，主要用來偵測遠端的障礙物，以提供給 D++演算法作為路經規劃的環境障礙物資訊。另外我們還使用了 2 個 GP2D120 紅外線感測器分別安裝在機器人前端左邊 45 度、與右邊 45 度之位置(如圖五十三所示)。GP2D120 之偵測範圍為 4cm ~ 30cm；主要為用來偵測左右近端的障礙物，以避免移動機器人因為編碼器與電子陀螺儀的誤差、或是輪胎打滑等因素，使機器人直線前進運動時偏斜，而與旁邊的障礙物碰撞。



圖五十三、紅外線感測器於本研究移動機器人平台之配置

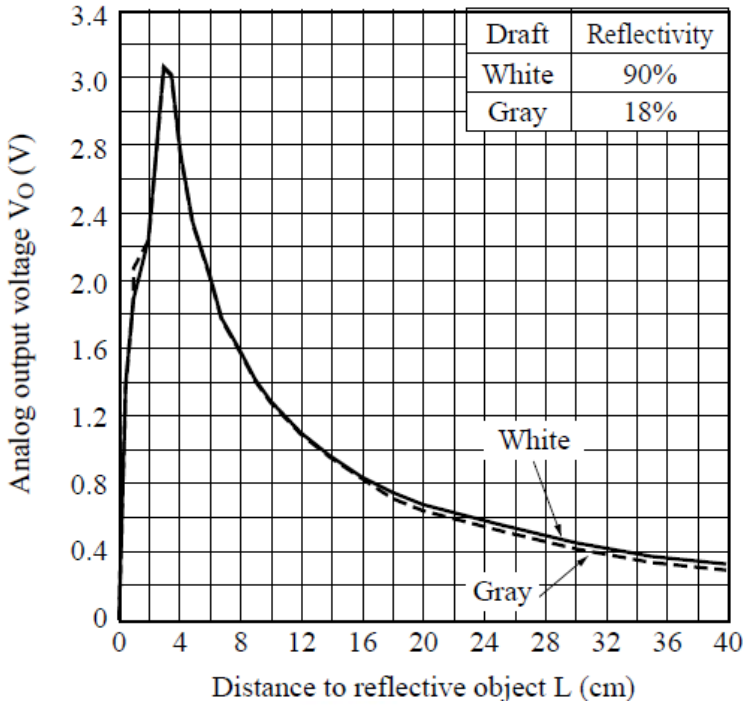
Sharp 的紅外線感測器是基於一個原理，三角測量原理。紅外發射器按照一定的角度發射紅外光束，當遇到物體以後，光束會反射回來，如圖 1 所示。反射回來 的紅外光線被 CCD 檢測器檢測到以後，會獲得一個偏移值 L，利用三角關係，在知道了發射角度 α，偏移距 L，中心矩 X，以及濾鏡的焦距 f 以後，感測器到物體的距離 D 就可以通過幾何關係計算出來(如圖五十四)。



圖五十四、紅外線感測器之原理圖

由圖五十四我們可以看到，當D的距離足夠近的時候，L值會相當大，超過CCD的探測範圍，這時，雖然物體很近，但是感測器反而看不到了。當物體距離D很大時，L值就會很小。這時CCD檢測器能否分辨得出這個很小的L 值成為關鍵，也就是說CCD檢測器的解析度決定能不能獲得足

夠精確的L 值。要檢測越是遠的物體，CCD檢測器的解析度要求就越高。Sharp的G系列感測器的輸出是非線性的。每個型號的輸出曲線都不同。所以在實際使用前，最好能對所使用的感測器進行一下校正。對每個型號的感測器創建一張曲線圖，以便在實際使用中獲得真實有效的測量資料(如圖五十五為GP2D120之輸出電壓與量測距離關係圖)。

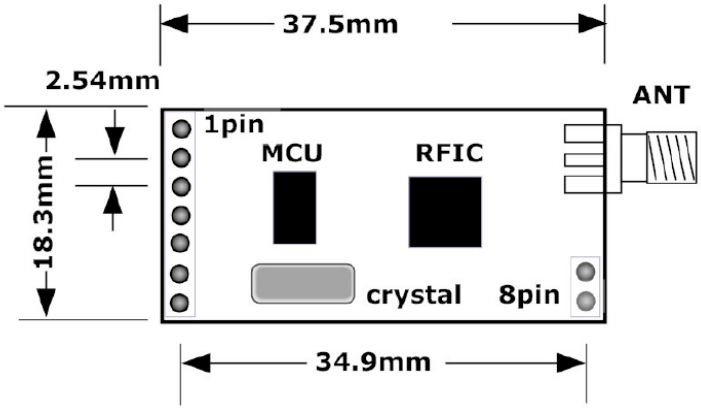


圖五十五、GP2D120 之輸出電壓與量測距離關係圖（來源：[59]）

3.5 無線電通訊模組

在本研究中，我們使用了 APC220 無線電通訊模組，來建立電腦與 Arduino 控制板的遠端溝通介面。APC220 無線電通訊模組包含一組的 APC220 通訊板、一組天線、一個 USB 轉接板(如圖五十六所示)。APC220 無線電通訊模組能夠有效率地傳輸任何大小的資料，而使用者無須編寫複雜的設置與傳輸程式，同時具有體積小、傳輸距離長，豐富而便利的軟體設置程式，使其能夠應用在非常廣泛的領域。

圖五十六、APC220 無線電通訊模組實體圖（來源：[56]）



圖五十七、APC220 無線電通訊模組尺寸圖（來源：[56]）

APC220 無線電通訊模組的特點:

1. 工作電壓：5V
2. 傳輸距離：1000 公尺（具體視環境而定）
3. 速率：1200/2400/4800/9600/19200/38400/57600 bps
4. 模組介面：UART/TTL
5. 工作頻率：418M-455MHz (1KHz 解析)
6. 高效率的編碼糾錯功能
7. 256 bytes 資料緩衝區
8. 適合大資料量傳輸
9. 內建 Watch sDog，保證長時間執行的可靠度
10. 線上修改設置

APC220 無線電通訊模組具有 9 個接腳，其接腳定義如所示：

表 7、APC220 無線電通訊模組接腳定義

接腳	定義	說明
1	GND	接地
2	Vcc	輸入電壓 3.3~5.5V
3	En	致能(空接或 1.6V 以上致能；0.5V 以下休眠)
4	Rx	Uart 輸入埠
5	Tx	Uart 輸出埠
6	Aux	UART 埠收發指示輸出(低電位接收；高電位輸出)
7	Set	參數設置，低電位有效。
8	NC	空腳位
9	NC	空腳位

另外 APC220 無線電通訊模組使用相當地靈活，可以根據使用者的需求設定不同的選項：

表 8、APC220 無線電通訊模組的參數設置表

參數	選項	預設
傳輸速率 （Series Rate）	1200、2400、4800、9600、19200、38400、57600 bps	9600bps
傳輸檢驗 （ Series Parity）	Disable、Even Parity、Odd Parity	Disable
收發頻率 (RF Frequency)	418MHz-455MHz (1KHz 解析)	434 MHz
空中傳輸速率 （Series Rate）	2400、4800、9600、19200 bps	9600bps
輸出功率(RF Power)	0-9 (9 為 20mw)	9(20mw)

APC220無線電通訊模組屬於半雙工傳輸模式，可以完成一對一，一對多的通訊。這二種方式首先需要設一個Master，其餘為Slave，所有模組都必須設置一個唯一的位址。通信的協調由Master控制，Master採用帶位址碼的發送資料或命令，所有Slave全部都接收，並將接收到的位址碼與本機位址碼比較，位址不同則將資料丟掉，不做回應，若位址碼相同，則將接收的資料傳送出去。模組網路必須保證在任何一個瞬間，同一個頻點通信網路中只有一個模組處於發送狀態，以免相互干擾。APC220無線電通訊模組可以設置多個頻道，所以可以在一個區域實現多個模組並存。考慮到空中傳輸的複雜性，無線資料傳輸方式固有的一些特點，應考慮以下幾個問題：

1. 無線通訊中資料的延遲

由於無線通訊發射端是從終端設備接收到一定數量的資料後，或等待一定的時間沒有新的資料才開始發射，無線通訊發射端到無線通訊接收端存在著幾十到幾百毫秒延遲(具體延遲是由串列傳輸速率，空中傳輸速率以及資料包的大小決定)，另外從無線通訊接收端到終端設備也需要一定的時間，但同樣的條件下延遲時間是固定的。

2. 資料流程量的控制

APC220無線電通訊模組雖然有256 bytes大容量緩衝區，但若串列傳輸速率大於等於空中傳輸速率，則存在資料流程量的問題，可能會出現資料超載而導致的資料丟失的現象。在這種情況下，終端設備要保證串列傳輸平均速率不大於60%空中傳輸速率。

3. 糾錯

APC220無線電通訊模組具有較強的抗干擾能力，在編碼上已經具有強大的糾檢錯能力。但在極端惡劣的條件下或場地的接收條件已處於APC220無線電通訊模組接收的臨界狀態，難免出現接收不到或封包遺失的狀況。

4. 天線的選擇

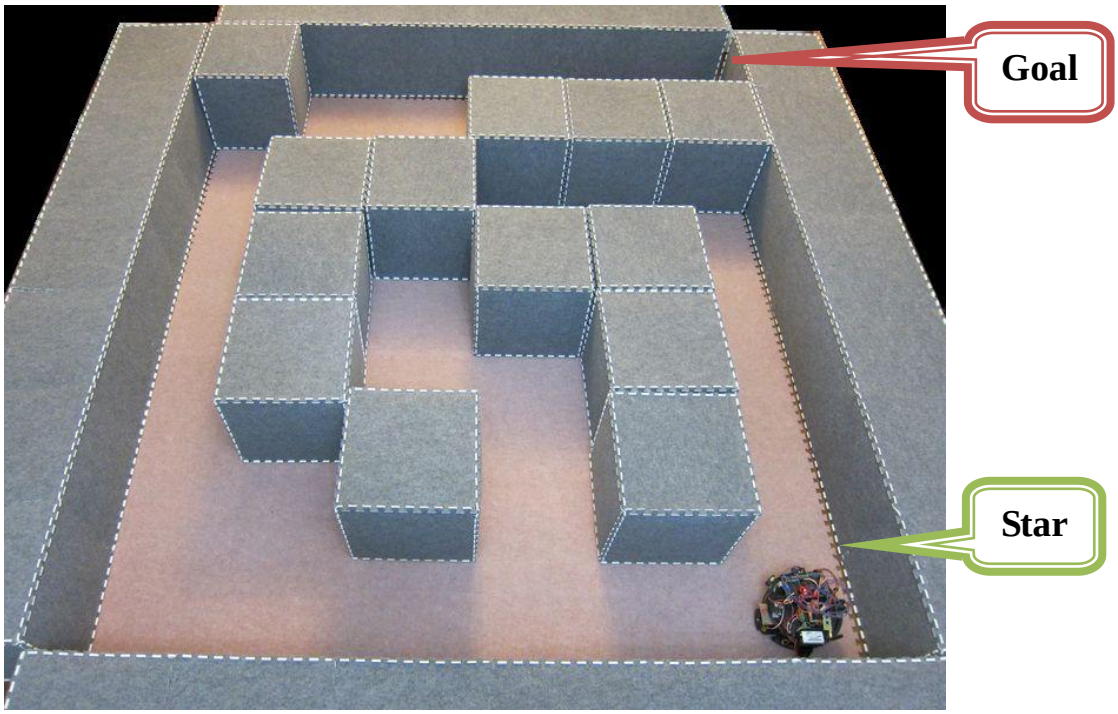
天線是通信系統的重要組成部分，其性能的好壞直接影響通信系統的指標，使用者在選擇天線時必須首先注重其性能。一般有兩個方面，第一選擇天線類型；第二選擇天線的電氣性能。選擇天線類型的意義是：所選天線的方向圖是否符合系統設計中電波覆蓋的要求；選擇天線電氣性能的要求是：選擇天線的頻率頻寬、增益、額定功率等電氣指標是否符合系統設計要求。

四、D++演算法應用於移動機器人之實驗結果

在本章節中，我們將上一章節所介紹的自製移動機器人平台，置放於自行架設的實驗環境空間中，並給予起點與終點設定，以驗證 D++演算法在實際的移動機器人應用中之效能，並根據實驗結果探討 D++演算法於實際應用上的優缺點。

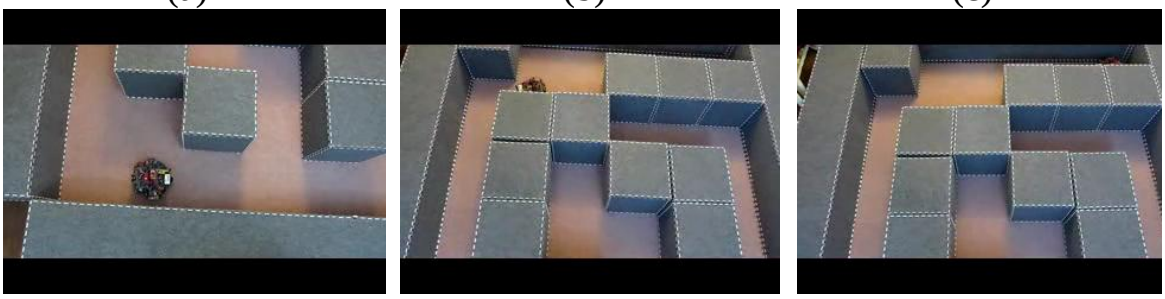
4.1 靜態環境

圖五十八為一個具迷宮地形的靜態環境，移動機器人需從地圖右下角尋找到地圖右上角的移動路徑。地圖中具有某些死路，讓機器人無法輕易地從起點直接走到終點。因此本實驗之目的為探討D++演算法對於應用在實際環境中，避免區域解的執行成效。圖五十九展示了機器人在此靜態環境的實際航行情形；而圖六十為電腦端顯示機器人航行的監視畫面。在這個例子中，移動機器人的偵測範圍為5個方格，每個方格為30×30公分。這個例子有利於我們了解D++演算法如何有效地應用在實際的移動機器人導航問題，且展示了D++演算法具有脫離或避開區域解的能力。



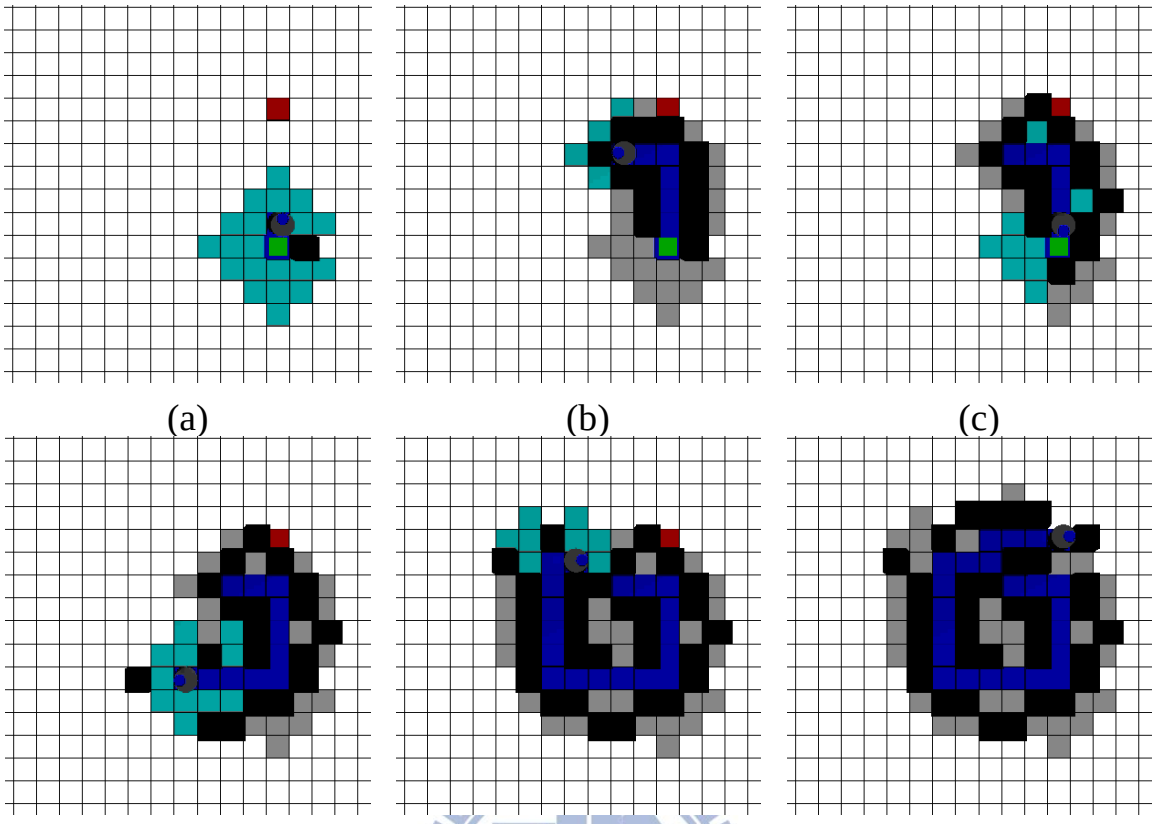
圖五十八、一個迷宮地形的靜態環境

(a) (b) (c)



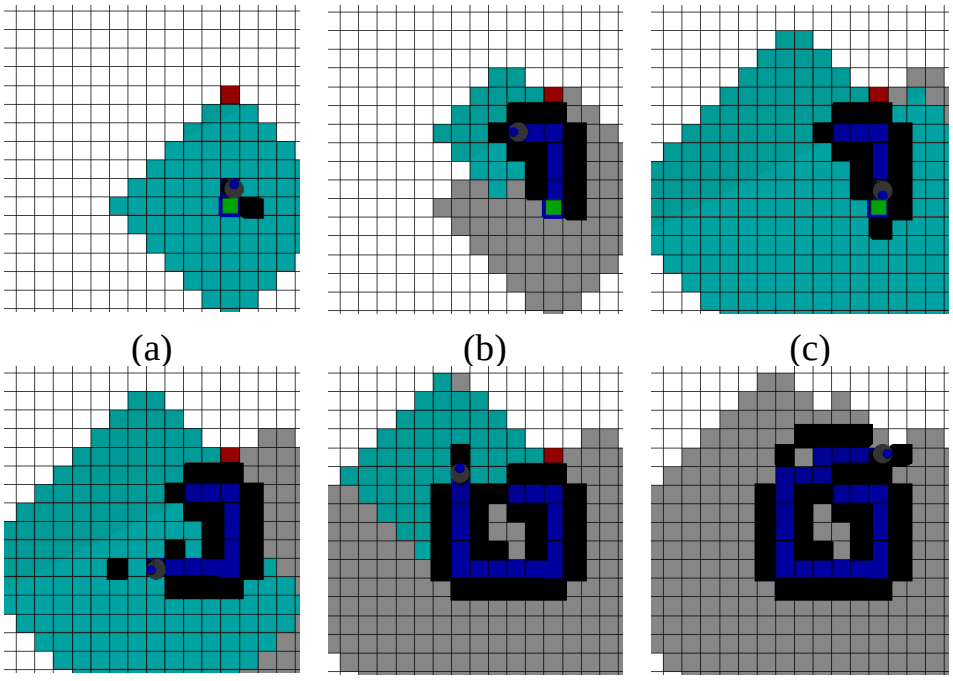
(d) (e) (f)

圖五十九、移動機器人航行在圖五十八之地形環境的實際情形（偵測範圍為 3 格）



圖六十、使用偵測範圍為 3 格的機器人航行實驗監視畫面

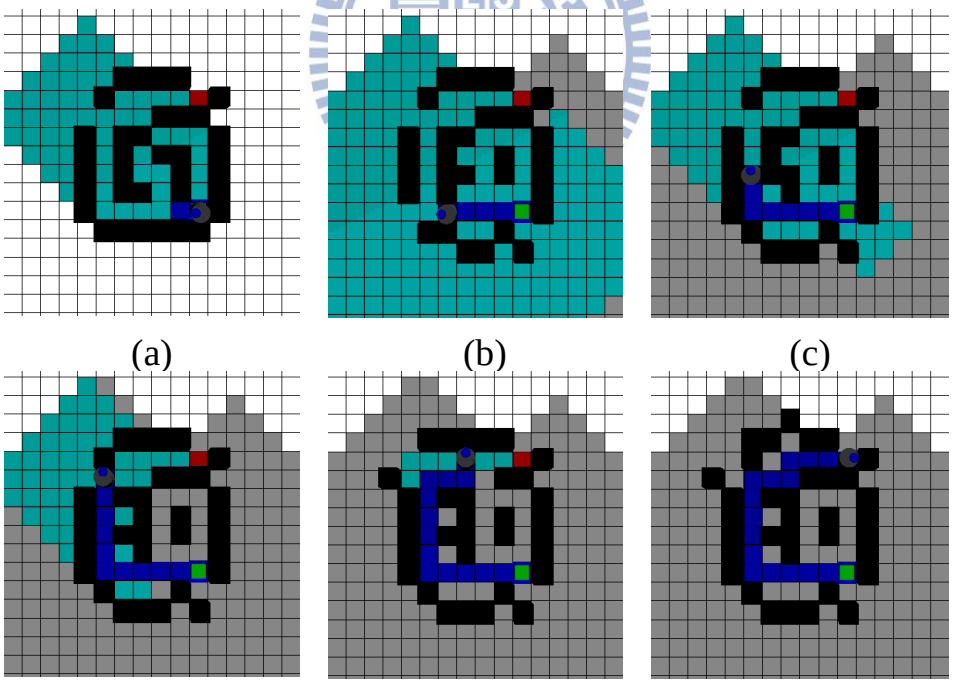
圖六十一展示了使用了偵測範圍為20格、在相同靜態環境（圖五十八）的移動機器人航行過程監視畫面； 圖六十一的例子雖然設定了20格的偵測範圍，但實際上因為移動機器人的紅外線感測器僅能偵測前左右的障礙物，且範圍僅有3~5格左右，因此並無法提供20格偵測範圍內的正確環境資訊。所以將圖六十一的結果與圖六十的結果作比較，我們可以看到移動機器人最後的航行路線並無不同。圖六十二為同樣使用了偵測範圍為20格、在相同靜態環境的移動機器人航行過程監視畫面；與圖六十一不同的是，這次環境資訊為已知的。將圖六十二的結果與圖六十一的結果作比較，我們可以看到圖六十二的航行路線便避開了一開始的區域解，直接往最短路徑的方向前進。



(a) (b) (c)

(d) (e) (f)

圖六十一、使用偵測範圍為 20 格的機器人航行實驗監視畫面



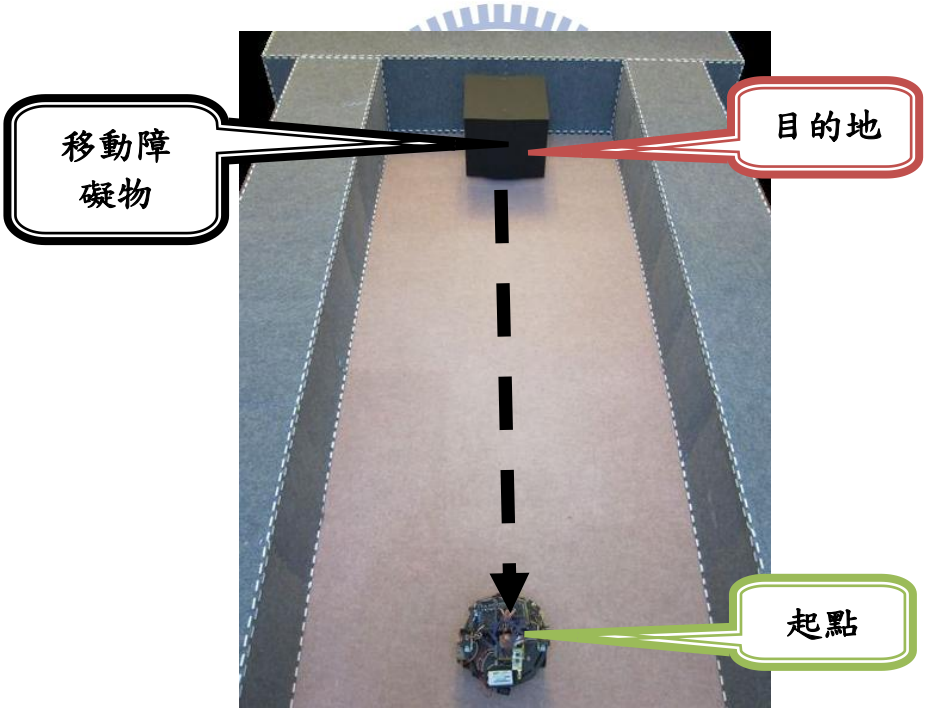
(a) (b) (c)

(d) (e) (f)

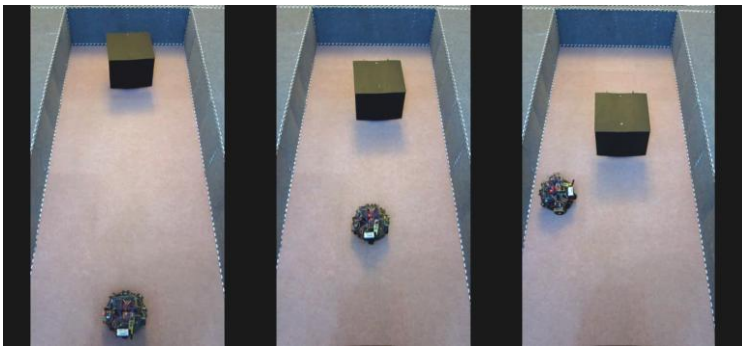
圖六十二、使用偵測範圍為 20 格、且已知環境資訊的機器人航行實驗監視畫面

4.2 動態環境

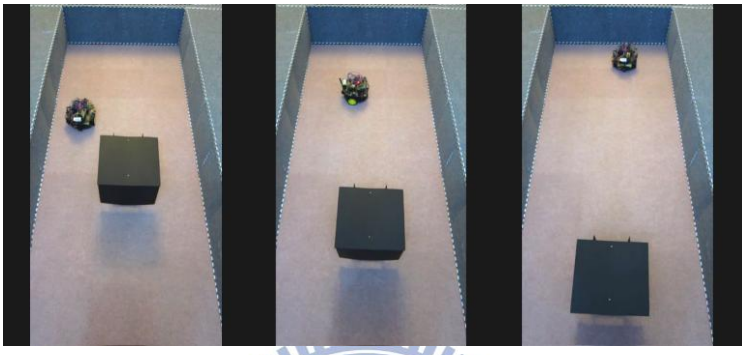
圖六十三~圖六十五展示了移動機器人航行在一個動態環境的實驗結果。在本實驗中機器人的偵測範圍設定為 5 格，且該環境因為具有一個向機器人靠近的移動物體，所以環境資訊為不斷變動的。由本實驗的結果我們可以看到，由於移動機器人應用 D++演算法，而 D++演算法具有極短周期的區域搜尋特性，因此移動機器人能不斷地更新環境資訊，並快速地進行相對應的動作。所以移動機器人在移動物體尚未靠近前，便先行轉向，繞開了移動物體已抵達目的地。故由此實驗可驗證 D++演算法由於具有處理動態環境的優秀能力，相較於一次性的全域路徑規劃法（如 Dijkstra、A*等）而言，更適合實際的移動機器人的應用上。



圖六十三、一個動態環境試驗(障礙物由上至下移動)

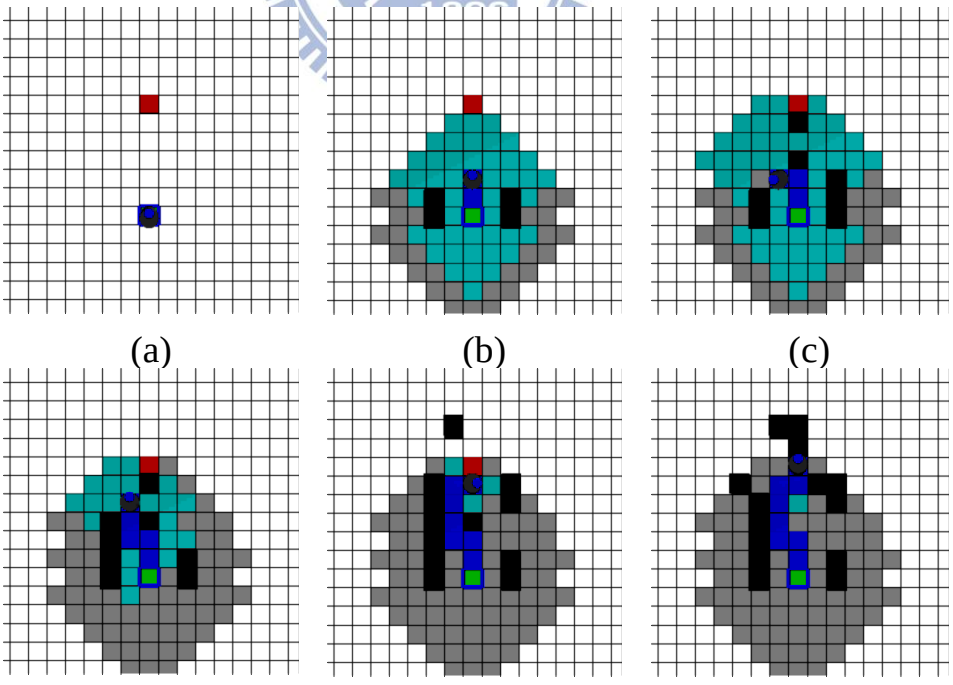


(a) (b) (c)



(d) (e) (f)

圖六十四、移動機器人航行在動態環境的實際情形（偵測範圍為 5 格）



(a) (b) (c)

(d) (e) (f)

圖六十五、使用偵測範圍為 5 格於動態環境的機器人航行實驗監視畫面

五、結論與建議

5.1 結論

本研究提出了一種動態路徑規劃法—D++演算法，此方法將原有的Dijkstra 演算法的搜尋空間限制於一特定大小(本研究稱之為「偵測範圍」)，故原來屬於全域搜尋法的 Dijkstra 演算法轉變成區域搜尋法。因為 D++演算法屬於區域路徑搜尋法，所以必須有暫定目標點。在本研究中我們訂定在每次區域搜尋時，搜尋範圍內離終點最近的點為暫定目標點（本研究稱之為「中繼點」)。藉著不斷進行區域搜尋並移動到新的中繼點，應用 D++演算法的機器人便會逐漸逼近目的地，最終抵達終點。因此 D++演算法可以在每次區域搜尋時，一邊更新環境資訊，一邊快速找出中繼點並移動至下一步；故此方法可以針對不斷變化的動態環境資訊快速地做出反應。而一般區域搜尋常碰到的區域最佳解問題，由於 D++演算法保有 Dijkstra 演算法的流程，故當機器人處在一個偵測範圍內無新節點的區域最佳解時，D++演算法可以暫時不斷擴大偵測範圍，直到新的節點被發現；故 D++演算法可以讓機器人避免陷入區域最佳解而卡住。總結來說 D++演算法結合了全域與區域路徑規劃優點，並適用於未知、大型、複雜的靜態或動態環境，所以 D++演算法為一種高泛用性的路徑規劃法。

本研究中，我們也自行建立了一組移動機器人平台，並將 D++演算法應用其上。該移動機器人平台具有紅外線感測器可偵測環境資訊。然而紅外線感測器屬直線型距離感測器，且感測距離有限，因此無法取得偵測範圍內完整而精確的環境資訊。所以 D++演算法應用在實際移動機器人上，偵測範圍大小的設定對最終路徑結果的影響（如圖六十與圖六十一)，並不如模擬實驗之結果如此明顯（如圖十四與圖十五)。因此 D++演算法若要應

用在實際的移動機器人，其效能與機器人的環境資訊感測器硬體性能有很直接且重要的關係。

除了應用了 D++演算法於移動機器人上，另外我們也成功應用了 D++演算法於舉重機器手臂的手端軌跡規劃上。使用 D++演算法可讓機器手臂立即規劃出可行移動路徑，並立即動作。若使用先前研究的方法（梯度最佳化法或傳統 Dijkstra 演算法），雖然可獲得最佳化結果，然而機器手臂必須計算相當長久的時間方能開始動作。故先前的方法較適用於計算一次後即不斷重複相同動作之應用（如工業製造等）。而 D++演算法較適用於需即時互動之應用（如醫療與家庭照護等）。

5.2 建議

雖然上述提到 D++演算法可用於未知、大型、複雜的動靜態環境，然而應用 D++演算法於上述環境中，需要具強大感測與移動能力的機器人。目前的機器人硬體受限於經費與時間的限制，尚無法完成上述的驗證目標。因此本研究後續的工作重點，應設法重新建立功能強大的機器人平台，如擁有強壯的結構、高速運算的控制器、高精度的編碼器、雷射掃描儀或 GPS…等。如此 D++演算法可望在不久將來有效地運用在實際的移動機器人導航上。

另外在舉重機器手臂的應用上，本研究中我們僅利用靜力學討論機器手臂各軸的受力情形，並未考慮到機器手臂運動時，加速度所產生的慣性力問題。這將是往後需要繼續深化的部分。此外目前也僅於用電腦軟體模擬結果，未來也需要進一步地將本研究的理論，實現在真實的機器手臂硬體上，並完成機器手臂實際避開障礙物的應用。

參考文獻

[1] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach. Upper Saddle River, N.J.: Prentice Hall. pp. 97–104, 2003.

[2] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik 1, pp. 269–271, 1959.

[3] P. E. Hart, N. J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* 4 (2): pp. 100–107, 1968.

[4] P. E. Hart, N. J. Nilsson, B. Raphael, “Correction to A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” SIGART Newsletter 37, pp. 28–29, 1972.

[5] R. Dechter; P. Judea, “Generalized best-first search strategies and the optimality of A*,” *Journal of the ACM* 32 (3): pp. 505–536, 1985.

[6] http://theory.stanford.edu/~amitp/GameProgramming/

[7] A. Stentz, “Optimal and Efficient Path-planning for Partially-Known Environments,” Proceedings of the International Conference on Robotics and Automation, pp. 3310–3317, 1994.

[8] A. Stentz. “The Focused D* Algorithm for Real-Time Replanning,” Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1652–1659, 1995.

[9] S. Koenig, M. Likhachev, “D* Lite,” In Proceedings of the Eighteenth National Conference on Artificial Intelligence, pp. 476-483, 2002.

[10] S. Koenig, M. Likhachev, D. Furcy. “Lifelong Planning A*,” Artificial Intelligence Journal, 155, (1-2), pp. 93-146, 2004.

[11] S. Koenig, Y. Smirnov and C. Tovey. Performance Bounds for Planning in Unknown Terrain. Artificial Intelligence Journal, 147, (1–2), pp. 253–279, 2003.

[12] G. Ramalingam, T. Reps, “An incremental algorithm for a generalization of the shortest-path problem,” Journal of Algorithms 21, pp. 267–305, 1996.

[13] D. Wooden, Graph-based Path Planning for Mobile Robots, Dissertation, Georgia Institute of Technology, 2006.

[14]S. Koenig, “A comparison of fast search methods for real-time situated agents,” In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, pp. 864–871, 2004.

[15] S. Koenig, M. Likhachev, “RealTime Adaptive A*,” AAMAS 2006 May 812, Hakodate, Japan, 2006

[16]O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” Robotics and Automation. Proceedings. IEEE International Conference on Volume 2, pp. 500 – 505, 1985.

[17] M. Çakir, E. Butun, Y. Kayman, "Effects of genetic algorithm parameters on trajectory planning for 6-DOF industrial robots", Industrial Robot: An International Journal, Vol. 33 Iss: 3, pp.205 – 215, 2006.

[18] Y. Koren, J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” Robotics and Automation, Proceedings of IEEE International Conference on 9-11, vol.2, pp. 1398 – 1404, 1991.

[19] S. Charifa, M. Bikdash, “Comparison of geometrical, kinematic, and dynamic performance of several potential field methods,” Southeastcon, 2009. SOUTHEASTCON '09, pp.18 – 23, 2009.

[20] P. Turennout, G. Honderd, L. J. Schelven, “Wall-following control of a mobile robot,”in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, pp. 280-285, 1992.

[21] X. Yun, K. C. Tan,"A wall-following method for escaping local minima in potential field based motion planning,"in *Proc. IEEE Int. Conf. Advanced Robotics*, pp. 421-426, July. 1997.

[22] 陳俊霖、李家儀、林浤偉、張永華，螞蟻演算法為基礎之移動式機器人最佳全域覆蓋路徑規劃，第17屆模糊理論及其應用研討會，2009年

[23] M. Dorigo, L.M. Gambardella, "Ant Colony System A Cooperative Learning Approach to The Traveling Salesman Problem," *IEEE Trans. on Evolutionary Computation*, Vol. 1, No. 1, pp. 53-66, 1997.

[24] M. Dorigo, V. Maniezzo, A.Colorni, "Ant System: Optimization by A Colony of Cooperating Agents," *IEEE Trans. System Man Cybernet .B*, Vol. 26, No. 1, pp. 29-42, 1996.

[25] M. Dorigo, L.M. Gambardella, "Ant Colony System A Cooperative Learning Approach to The Traveling Salesman Problem," *IEEE Trans. on Evolutionary Computation*, Vol. 1, No. 1, pp. 53-66, 1997.

[26] http://www.thefullwiki.org/Ant_colony_algorithm

[27] 蘇木春、張孝德，機器學習：類神經網路、模糊系統以及基因演算法則，全華科技股份有限公司，民國 91 年

[28] Y. Hu, S. X. Yang, "A Knowledge Based Genetic Algorithm for Path Planning of a Mobile Robot," in Proc. IEEE International Conf. Robotic and Automation, New Orleans, USA, pp. 4350-4356, 2004.

[29] L. Lei, H. Wang, Q. Wu, "Improved Genetic Algorithm Based Path Planning of Mobile Robot under Dynamic Unknown Environment," in Proc. IEEE International Conf. Mechatronics and Automation, Luoyang, China, pp. 1728-1732, 2006.

[30] J. Lu, D. Yang, "Path Planning Based on Double-layer Genetic Algorithm," in Proc. 3th IEEE International Conf. Natural Computation, Hainan, China, 2007.

[31] H. Mahjoubi, F. Bahramin, C. Lucas, “Path Planning in an Environment with Static and Dynamic Obstacles Using Genetic Algorithm: A Simplified Search Space Approach,” In Proc. of IEEE Cong. Evolutionary Computation, Vancouver , Canada, pp. 2483-2488, 2006.

[32] A. Ismail, A. Sheta, M. Al-Weshah, “A Mobile Robot Path-planning Using Genetic Algorithm in Static Environment,” Journal of Computer Science, v4, pp. 341-344, 2008.

[33] H. J. Ritter, T. M. Martinetz, K. J. Schulten, “Topology-conserving maps for learning visuo-motor-coordination,” Neural Netw., vol. 2, no. 3, pp. 159–189, 1989.

[34] P. Gaudiano, E. Zalama, J. L. Coronado, “An unsupervised neural network for low-level control of a mobile robot: Noise resistance, stability, and hardware implementation,” IEEE Trans. Syst., Man, Cybern. B. Cybern, vol. 26, no. 3, pp. 485–496, 1996.

[35] E. Zalama, P. Gaudiano, J. L. Coronado, “A real-time, unsupervised neural network for the low-level control of a mobile robot in a nonstationary environment,” Neural Netw., Vol. 8, no. 1, pp. 103–123, 1995.

[36] L. Li, H. Ö ˇgmen, “Visually guidedmotor control: Adaptive sensorimotor mapping with on-line visual-error correction,” in Proc. World Congr. Neural Networks, San Diego, CA, pp. 127–134, 1994.

[37] S. Yang, M. Meng, “Neural Network Approaches to Dynamic Collision-Free Trajectory Generation,” IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol.31, No.3, pp. 302-318, 2001.

[38]A.R. Willms, S.X. Yang, “An efficient dynamic system for real-time robot-path-planning,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, Vol. 36, Issue 4, pp.755-766 , 2006.

[39] K.S. Senthilkumar, K.K. Bharadwaj, “Hybrid Genetic-Fuzzy Approach to Autonomous Mobile Robot,” IEEE International Conference of Technologies for Practical Robot Applications (TePRA), pp. 29–34, 2009.

[40] I. Hassanzadeh, S.M. Sadigh, “Path-planning for a mobile robot using fuzzy logic controller tuned by GA,” 6th International Symposium Mechatronics and its Applications (ISMA), pp.1–5, 2009.

[41]Y. Fu and S. Y. T. Lang, “Fuzzy Logic Based Mobile Robot Area Filling with Vision System for Indoor Environments”, *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, pp.326-331, 1999.

[42]P.Y. Cheng, D.C. Liu, “The Shortest Path Planning for Robot on a 3D Obstacle,” Proceedings of International Conference of the Automation, 2003.

[43]M. Kim, N. Y. Chong, W. Yu, “Fusion of direction sensing RFID and sonar for mobile robot docking,” Automation Science and Engineering, CASE 2008, pp.709 – 714, 2008.

[44] A. K. Ray, L. Behera, M. Jamshidi, “Sonar-Based Rover Navigation for Single or Multiple Platforms: Forward Safe Path and Target Switching Approach,” IEEE Systems Journal, Vol. 2, No. 2, pp. 258-272, 2008.

[45]楊雅兆，使用超音波感測之自走車避障實務設計，中原大學機械工程學系碩士論文，民國93年7月

[46] P.Y. Cheng, P.J. Chen, “The D++ Algorithm Real-Time and Collision-Free Path-Planning for Mobile Robot”, The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), Taipei, 2010.

[47]P.Y. Cheng, P.J. Chen, “A Real Time Path Planning for Weightlifting Manipulator Using D++ Algorithm”, The Fisrt IFToMM Asian Conference on Mechanism and Machine Science, Taipei, 2010.

[48]C.Y. Wang, W. Timoszyk, J. Bobrow, “Payload Maximization for Open Chained Manipulators: Finding Weightlifting Motions for a Puma 762 Robot,” IEEE Transactions on Robotics and Automation, Vol.17, No. 2, 2001.

[49] M. Rosenstein, A. Barto, “Robot Weightlifting by Direct Policy Search,” In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Vol. 2, pp. 839-844, 2001.

[50]P.Y. Cheng, C.Y. Chen, “Path planning and dynamic simulation of weightlifting robot manipulator,” Advanced Robotics and Its Social Impacts, 2007. ARSO 2007. IEEE Workshop on Digital Object, 10.1109/ARSO.2007.4531426, pp. 1–6, 2007.

[51]鄭榮煌，使用光流影像感測的全方向運動平台之空間軌跡追蹤系統設計，國立交通大學電機與控制工程系所碩士論文，民國94年7月

[52] http://www.arduino.cc

[53]蔡宗成，小型馬達驅動**IC**簡介與應用，元智大學機械工程學系最佳化設計實驗室文獻，民國90年9月

[54] http://www.datasheetdir.com/HD74LS14+Inverters-Gates

[55] http://www.elecfans.com/baike/bandaoti/jichuzhishi/20100308182701.html

[56] http://www.dfrobot.com

[57] http://www.datasheetcatalog.com

[58] http://www.sparkfun.com/products/9059

[59] http://www.robot-electronics.co.uk/datasheets/gp2d120.pdf

[60] K.H. Ang, G.C.Y. Chong, Y. Li, “PID control system analysis, design, and technology,” IEEE Trans Control Systems Tech 13(4), pp. 559-576, 2005.

[61] Y. Li, K.H. Ang, G.C.Y. Chong, “PID control system analysis and design,” IEEE Control Systems Magazine 26(1), pp. 32-41, 2006.

附錄一、D++演算法程式碼

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;

namespace ShortPathSearch
{
    public class Node{
        public List<float> Cost = new List<float>();
        public List<bool> Obstacle = new List<bool>();
        public List<int> Father = new List<float>();
        public List<int[]> Axis = new List<int[]>();
        public int[] AxisRange;
        public int Number;
        public int AxisAmount;
        public int NodeRange;
        public int[] Loop;
        public int[] LoopAmount;
        public int[] Step;
        public int BestNumber;
        public float BestCost;
        public int[] BestAxis;
        public int[] TempAxis;
        public int StartNumber;
        public int GoalNumber;
        public int[] StartAxis;
        public int[] GoalAxis;

        public Node(int axis_amount, int[] max_value, int[] min_value){

            AxisAmount = axis_amount;
            AxisRange = new int[AxisAmount];
            LoopAmount = new int[AxisAmount];
            Loop = new int[AxisAmount];
            Step = new int[AxisAmount];
            BestAxis = new int[AxisAmount];
            TempAxis = new int[AxisAmount];

            NodeRange = 1;
            for(int i=0; i<AxisAmount; i++){
                //計算各軸節點之範圍
                AxisRange[i] = max_value[i] - min_value[i] + 1;
                NodeRange = NodeRange * AxisRange[i];
                Loop[i] = 0;
                Step[i] = 0;
                //計算各軸排列組合之循環次數
```

```
                if(i==0){
                    LoopAmount[i] = 1;
                }
                else{
                    LoopAmount[i] = LoopAmount[i-1] * AxisRange[i-1];
                }
            }

            //排列組合節點刻度表
            for(int i=0; i<NodeRange; i++){
                for(int j=0; j<AxisAmount; j++){
                    Axis.Add(new int[AxisAmount]);
                    Axis[i][j] = min_value[j] + Step[j];
                    Loop[j] = Loop[j] + 1;
                    if(Loop[j] >= LoopAmount[j]){
                        Loop[j] = 0;
                        Step[j] = Step[j] + 1;
                        if(min_value[j] + Step[j] > max_value[j]){
                            Step[j] = 0;
                        }
                    }
                }
            }
        }

        //清除節點資料
        public void NodeDataClear(){
            Cost.Clear();
            Obstacle.Clear();
            Father.Clear();
            for(int i=0 ; i<NodeRange; i++){
                Cost.Add(0);
                Obstacle.Add(false);
                Father.Add(0);
            }
        }
    }

    public class PathSearch
    {
        public bool SearchSuccess;
        public int[] MaxValue;
        public int[] MinValue;
        public int SearchStep;

        List<int> OpenList = new List<int>();
        List<int> CloseList = new List<int>();

        public List<float[]> PathList = new List<float[]>();
```

```
        Node Node;

        //初始化
        public void Initialize(int axis_amount, int[] start_axis, int[] goal_axis, int[]
max_value, int[] min_value, int step)
        {
            Node = new Node(axis_amount, max_value, min_value);
            MaxValue = new int[Node.AxisAmount];
            MinValue = new int[Node.AxisAmount];
            BestNodeAxis = new int[Node.AxisAmount];
            SearchStep =step;
            SearchSuccess = false;

            for(int i=0; i<Node.AxisAmount; i++){
                Node.StartAxis[i] = start_axis[i];
                Node.GoalAxis[i] = goal_axis[i];
            }
        }

        //計算節點的編號值
        public int GetNodeNumber(int[] axis){

            int node_number = 0;
            for(int i=0; i<Node.AxisAmount; i++){
                node_number = node_number + (axis[i] - MinValue[i]) *
                (int)Math.Pow(Node.AxisRange[i],i);
            }

            return node_number;
        }

        //開始搜尋
        public void Search(){

            SearchInitialize();
            SearchSuccess = false;
            for(;;){

                //從 OpenList 表中取得一個 CostFunction 最小的節點
                FindBestNode();

                //將該節點從 OpenList 表中移除並加入 CloseList 表中
                OpenList.Remove(Node.BestNumber);
                CloseList.Add(Node.BestNumber);

                //如果該節點為終點則退出搜索
                if(Node.BestNumber == Node.GoalNumber){
```

```
            SearchSuccess = true;
            break;
        }

        //否則生成子節點
        for(int i=0; i<Node.AxisAmount; i++){
            Node.Loop[i] = 0;
            Node.Step[i] = -SearchStep;
            //計算各軸排列組合之循環次數
            if(i==0){
                Node.LoopAmount[i] = 1;
            }
            else{
                Node.LoopAmount[i] = Node.LoopAmount[i-1] * 2;
            }
        }
        for(int i=0; i<Math.Pow(2,Node.AxisAmount); i++){
            for(int j=0; j<Node.AxisAmount; j++){
                Node.TempAxis[j] = Node.BestAxis[j] + Node.Step[j];
                Node.Number = GetNodeNumber(Node.TempAxis);
                if( Node.Number != Node.BestNumber){
                    ChildNodeMake();
                }
                Node.Loop[j] = Node.Loop[j] + 1;
                if(Node.Loop[j] >= Node.LoopAmount[j]){
                    Node.Loop[j] = 0;
                    Node.Step[j] = Node.Step[j] + SearchStep;
                    if(Node.Step[j] > SearchStep){
                        Node.Step[j] = -SearchStep;
                    }
                }
            }
        }

        //如果 Open 表裡無節點,代表搜尋無解,則退出搜索
        if(OpenList.Count == 0){
            break;
        }
    }

    //取得路徑上的節點
    if(SearchSuccess == true){
        GetPath();
    }
    else{
        MessageBox.Show("No Solution!!");
    }
    MessageBox.Show("Search Finish!");
```

```
}

//搜尋初始化
void SearchInitialize()
{
	//清除 OpenList 與 CloseList 表
	OpenList.Clear();
	CloseList.Clear();

	//清除節點資料
	Node.NodeDataClear();

	//生成初始資料
	Node.StartNumber = GetNodeNumber(Node.StartAxis);
	Node.BestNumber = Node.StartNumber;
	OpenList.Add(Node.BestNumber);

	//生成目標資料
	Node.GoalNumber    = GetNodeNumber(Node.GoalAxis);
}

//找尋最佳節點
void FindBestNode(){
	for(int i=0; i<OpenList.Count; i++){
		if(i == 0 || Node.Cost[OpenList[i]] < Node.BestCost){
			Node.BestNumber = OpenList[i];
			Node.BestCost = Node.Cost[Node.BestNumber];
			for(int j=0; j<Node.AxisAmount; j++){
				Node.BestAxis[j] = Node.Axis[Node.BestNumber][j];
			}
		}
	}
}

//產生子節點
void ChildNodeMake(){
	float new_node_cost;

	//子節點的 Cost 值
	new_node_cost = GetCost(Node.Number);

	//檢查子節點是否在 Open 表中
	if(OpenList.Contains(Node.Number) == true){
		//在 Open 表中,若子節點的新 Cost 值比較小
		if(new_node_cost < Node.Cost[Node.Number]){
			//則更新子節點的資料
			Node.Cost[Node.Number] = new_node_cost;
```

```
                Node.Father[Node.Number] = Node.BestNumber;
            }
        }

        //檢查子節點是否在 Close 表中
        else if(CloseList.Contains(Node.Number) == true){
            //在 Close 表中,若子節點的新 Cost 值比較小
            if(new_node_cost < Node.Cost[Node.Number]){
                //則更新子節點的資料
                Node.Cost[Node.Number] = new_node_cost;
                Node.Father[Node.Number] = Node.BestNumber;

                //將節點重新加入 Open 表中
                OpenList.Add(Node.Number);

                //將節點從 Close 表中移除
                CloseList.Remove(Node.Number);
            }
        }

        //若不在 Open 表中也不在 Close 表中
        else{
            //增加子節點的資料
            Node.Cost[Node.Number] = new_node_cost;
            Node.Father[Node.Number] = Node.BestNumber;

            //將節點加入 Open 表中
            OpenList.Add(Node.Number);
        }
    }

    //計算節點 Cost 值
    float GetCost(int node_number){
        float cost;
        return cost;
    }

    //產生路徑
    void GetPath(){

        //從終點回溯父節點到起點
        PathList.Clear();
        PathList.Insert(0, new float[AxisAmount]);
        for(int i=0 ; i<AxisAmount ; i++){
            PathList[0][i] = Node.GoalAxis[i];
        }
        Node.Number = Node.GoalNumber;
```

```
                for(;;){
                    PathList.Insert(0, new float[AxisAmount]);
                    for(int i=0 ; i<AxisAmount ; i++){
                        PathList[0][i] = Node.Axis[Node.Number][i];
                    }
                    //如果節點位於起點表就退出
                    if(Node.Number == Node.StartNumber){
                        break;
                    }
                    Node.Number = Node.Father[Node.Number];
                }
            }
        }
    }
```

附錄二、Arduino 開發板程式碼

```
#include <FlexiTimer2.h>

int incomingByte;
int RdyGetByte;

int LftIRLimit, RhtIRLimit, CenIRLimit;
volatile int LftIRVal, RhtIRVal, CenIRVal;
const int FarIRSensorLftPin = 0;
const int FarIRSensorCenPin = 1;
const int FarIRSensorRhtPin = 2;
const int CloseIRSensorLftPin = 3;
const int CloseIRSensorRhtPin = 4;

const int GSSensorPin = 5;
int GSSensorValError, InitialGSSensorVal;

const int EncLftPin = 2;
const int EncRhtPin = 3;
volatile int EncLft, EncRht;
boolean EncInterruptStop;

int Turn90, Turn180, TurnDir;
int StraightStep, Step, HalfStep;
volatile int TempStep;

const int PWMLft = 6;
const int DIRLft = 7;
const int PWMRht = 5;
const int DIRRht = 4;

int RunMode;
int MotorSpeed;
int Max_Speed, Min_Speed;
int LftMotorSpeed, RhtMotorSpeed;
int Acc;

float Kp, Kd, Ki;
int pError, dError, iError, Last_pError, Pre_pError;

//--------------------------------------------------------------------------
void setup(){
      Serial.begin(9600);

      pinMode(FarIRSensorLftPin, INPUT);
      pinMode(FarIRSensorCenPin, INPUT);
```

```
	pinMode(FarIRSensorRhtPin, INPUT);

	pinMode(CloseIRSensorLftPin, INPUT);
	//pinMode(CloseIRSensorCenPin, INPUT);
	pinMode(CloseIRSensorRhtPin, INPUT);

	pinMode(DIRLft, OUTPUT);
	pinMode(DIRRht, OUTPUT);

	attachInterrupt(0, LtfWheelEnc, CHANGE); //init the interrupt mode for pin 2
	attachInterrupt(1, RhtWheelEnc, CHANGE); //init the interrupt mode for pin 3

	EncLft = 0;
	EncRht = 0;
	RdyGetByte = 4;
	RunMode = 0;
	EncInterruptStop = false;

	InitialGSSensorVal = 0;
	for(int i=0; i<10; i++){
		InitialGSSensorVal =    InitialGSSensorVal + analogRead(GSSensorPin);
	}
	InitialGSSensorVal = InitialGSSensorVal / 10;
}
//-------------------------------------------------------------------------
void LtfWheelEnc(){
	if(EncInterruptStop == false){
		EncLft = EncLft + 1;
	}
}
//-------------------------------------------------------------------------
void RhtWheelEnc(){
	if(EncInterruptStop == false){
		EncRht = EncRht + 1;
	}
}
//-------------------------------------------------------------------------
void Timer(){
	if(RunMode == 1){
		if(EncLft >= Step || analogRead(FarIRSensorCenPin) >= CenIRLimit){
			RunMode = 0;
			analogWrite(PWMLft, 0);
			analogWrite(PWMRht, 0);
			Serial.write(RdyGetByte);
			FlexiTimer2::stop();
			if(EncLft >= Step){
				SendStatus(EncLft);
			}
			else{
```

```
            SendStatus(RdyGetByte);
        }
    }
    else{
        LftIRVal = analogRead(CloseIRSensorLftPin);
        RhtIRVal = analogRead(CloseIRSensorRhtPin);
        while(LftIRVal > LftIRLimit || RhtIRVal > RhtIRLimit){
            EncInterruptStop = true;
            if(LftIRVal > RhtIRVal){
                digitalWrite(DIRLft, HIGH);
                digitalWrite(DIRRht, LOW);
            }
            else{
                digitalWrite(DIRLft, LOW);
                digitalWrite(DIRRht, HIGH);
            }
            analogWrite(PWMLft, Min_Speed);
            analogWrite(PWMRht, Min_Speed);
            LftIRVal = analogRead(CloseIRSensorLftPin);
            RhtIRVal = analogRead(CloseIRSensorRhtPin);
        }

        EncInterruptStop = false;
        pError = analogRead(GSSensorPin) - InitialGSSensorVal;
        iError = iError + pError;
        dError = pError - Last_pError;
        Last_pError = pError;

        if(MotorSpeed < Max_Speed && EncLft < HalfStep){
            MotorSpeed = MotorSpeed + Acc;
        }
        else if(MotorSpeed > Min_Speed && EncLft > HalfStep){
            MotorSpeed = MotorSpeed - Acc;
        }
        LftMotorSpeed = int(MotorSpeed - Kp * pError - Ki * iError - Kd *
                            dError);
        RhtMotorSpeed = int(MotorSpeed + Kp * pError + Ki * iError + Kd *
                            dError);
        if(LftMotorSpeed > Max_Speed){
            LftMotorSpeed = Max_Speed;
        }
        else if(LftMotorSpeed < Min_Speed){
            LftMotorSpeed = Min_Speed;
        }
        if(RhtMotorSpeed > Max_Speed){
            RhtMotorSpeed = Max_Speed;
        }
        else if(RhtMotorSpeed < Min_Speed){
            RhtMotorSpeed = Min_Speed;
```

```
            }
            digitalWrite(DIRLft, HIGH);
            digitalWrite(DIRRht, HIGH);
            analogWrite(PWMLft, LftMotorSpeed);
            analogWrite(PWMRht, RhtMotorSpeed);
        }
    }
    else if(RunMode == 2){
        GSSensorValError = analogRead(GSSensorPin) - InitialGSSensorVal;
        if(abs(GSSensorValError) > 10){
            TempStep = TempStep + GSSensorValError /10;
        }
        pError = Step - TempStep;
        if(abs(pError) <= 20){
            RunMode = 0;
            analogWrite(PWMLft, 0);
            analogWrite(PWMRht, 0);
            Serial.write(RdyGetByte);
            FlexiTimer2::stop();
        }
        else{
            iError = iError + pError;
            dError = pError - Last_pError;
            Last_pError = pError;
            MotorSpeed = abs(int(Kp * pError + Ki * iError + Kd * dError));

            if(pError > 0){
                digitalWrite(DIRLft, HIGH);
                digitalWrite(DIRRht, LOW);
            }
            else{
                digitalWrite(DIRLft, LOW);
                digitalWrite(DIRRht, HIGH);
            }

            if(MotorSpeed > Max_Speed){
                MotorSpeed = Max_Speed;
            }

            analogWrite(PWMLft, MotorSpeed);
            analogWrite(PWMRht, MotorSpeed);
        }
    }
}
//--------------------------------------------------------------------------
void Move(){
    digitalWrite(DIRLft, HIGH);
    digitalWrite(DIRRht, HIGH);
```

```
        Step = StraightStep;
        HalfStep = Step / 2;
        EncLft = 0;
        EncRht = 0;
        iError = 0;
        Last_pError = 0;
        MotorSpeed = Min_Speed;
        RunMode = 1;
        FlexiTimer2::set(10, Timer); // 10ms period
        FlexiTimer2::start();
}
//------------------------------------------------------------------------
void Turn(int dir, int turn_angle){

        if(turn_angle == 1){
                Step = Turn90;
        }
        else{
                Step = Turn180;
        }

        TurnDir = dir;
        if(TurnDir == 1){
                Step = -Step;
        }

        TempStep = 0;
        iError = 0;
        Last_pError = 0;
        RunMode = 2;
        FlexiTimer2::set(25, Timer); // 50ms period
        FlexiTimer2::start();
}
//------------------------------------------------------------------------
void SendStatus(int data){
        byte data_h, data_l;

        data_h = data / 256;
        data_l = data % 256;
        Serial.write(data_h);
        Serial.write(data_l);

        return;
}
//------------------------------------------------------------------------
void IRSense(int far_lft_pin, int far_cen_pin, int far_rht_pin, int clo_lft_pin, int
                clo_rht_pin)
{
        int n = 6;
```

```
        int m = 50;
        int i;
        int ir_sensor_val[n];
        int check_sum = 0;
        int data_H, data_L;

        for(i = 0; i < n ; i++){
                ir_sensor_val[i] = 0;
        }

        for(i = 0; i < m ; i++){
                ir_sensor_val[0] = ir_sensor_val[0] + analogRead(far_lft_pin);
                ir_sensor_val[1] = ir_sensor_val[1] + analogRead(far_cen_pin);
                ir_sensor_val[2] = ir_sensor_val[2] + analogRead(far_rht_pin);
                ir_sensor_val[3] = ir_sensor_val[3] + analogRead(clo_lft_pin);
                ir_sensor_val[4] = ir_sensor_val[4];
                ir_sensor_val[5] = ir_sensor_val[5] + analogRead(clo_rht_pin);
        }

        for(i = 0; i < n ; i++){
                ir_sensor_val[i] = ir_sensor_val[i] / m;
                data_H = ir_sensor_val[i] / 256;
                data_L = ir_sensor_val[i] % 256;
                Serial.write(data_H);
                Serial.write(data_L);
                check_sum = check_sum + ir_sensor_val[i];
        }
        check_sum = check_sum % 256;
        Serial.write(check_sum);

    return;
}
//--------------------------------------------------------------------------
void GetGyroscope(){
        int gs_sensor_val;
        int data_h, data_l;

        gs_sensor_val = analogRead(GSSensorPin);
        data_h = gs_sensor_val / 256;
        data_l = gs_sensor_val % 256;
        Serial.write(data_h);
        Serial.write(data_l);
        return;
}
//--------------------------------------------------------------------------
void DataUpdate(){
        int data_number = 12;
        int recieve_data[data_number];
        int check_sum = 0;
```

```
        int i;
        int feed_back;

        for(i=0; i<data_number; i++){
                while(Serial.available() <= 0){};
                recieve_data[i] = Serial.read();
                check_sum = check_sum + recieve_data[i];
        }
        check_sum = check_sum % 256;

        StraightStep = recieve_data[0];
        Turn90 = recieve_data[1] * 10;
        Turn180 = recieve_data[2] * 10;
        Kp = recieve_data[3] * 1.0 / 100;
        Kd = recieve_data[4] * 1.0 / 100;
        Ki = recieve_data[5] * 1.0 / 100;
        LftIRLimit = recieve_data[6] * 10;
        CenIRLimit = recieve_data[7] * 10;
        RhtIRLimit = recieve_data[8] * 10;
        Max_Speed = recieve_data[9];
        Min_Speed = recieve_data[10];
        Acc = recieve_data[11];

        Serial.write(check_sum);
        return;
}
//-------------------------------------------------------------------
void loop(){
        int dir, turn_angle;

        if(Serial.available() > 0){
                incomingByte = Serial.read();
                switch(incomingByte){
                        case 67://'C', Clear
                                EncLft = 0;
                                break;
                        case 69://'E', Get Encoder Value
                                SendStatus(EncLft);
                                break;
                        case 70://'F', Forward
                                Move();
                                break;
                        case 71://'G', Forward
                                GetGyroscope();
                                break;
                        case 83://'S', Sense
                                IRSense(FarIRSensorLftPin, FarIRSensorCenPin, FarIRSensorRhtPin,
                                        CloseIRSensorLftPin, CloseIRSensorRhtPin);
                                break;
```

```
                case 84://'T', Turn
                    while((Serial.available() <= 0)){};
                    dir = Serial.read();
                    while((Serial.available() <= 0)){};
                    turn_angle = Serial.read();
                    Turn(dir, turn_angle);
                    break;
                case 85://'U', Update
                    DataUpdate();
                    break;
                default:
                    break;
            }
        }
}
```