國立臺灣大學工學院應用力學研究所

碩士論文

Graduate Institute of Applied Mechanics
College of Engineering
National Taiwan University

Master Thesis

改良式 A*演算法於動態環境路徑規劃與避障之應用

Path planning with structure modified A* algorithm in dynamic environment.

林宗佑

Tsung-You Lin

指導教授：王立昇 博士

Advisor: Li-Sheng Wang, Ph.D

中華民國 107 年 7 月

July,2018

doi:10.6342/NTU201802627

誌謝

誠摯的感謝指導老師王立昇博士兩年來的悉心指導，也特別感謝口試委員張帆人博士、卓大靖博士、王和盛博士給予我寶貴的建議與回饋。王老師的指導讓我在專業領域上受益良多，並且在面對研究的困境時也能夠耐心地尋找突破與解決方法。

研究所生涯中，感謝實驗室學長、同學與學弟的陪伴與照顧。特別感謝劉禮榮同學與吳泓霆在研究時給予的協助、建議與督促。能夠與實驗室同學一起閒聊與討論，在困難時互相扶持，讓我在研究所的路上更有勇氣披荊斬棘。另外，特別感謝電機所同學翁祖彬、鄭哲安與機械系學弟魏程裕還有我女朋友何宜蕙在我研究所時經常找我聊天吃飯，傾聽我研究與生活上遇到的困難。

最後，感謝我的父母與妹妹在我研究所的時候經常得關心並且提供生活上的協助與精神上的鼓勵。謝謝你們一路上的支持與鼓勵，是我能夠完成學業的最大助力。

doi:10.6342/NTU201802627

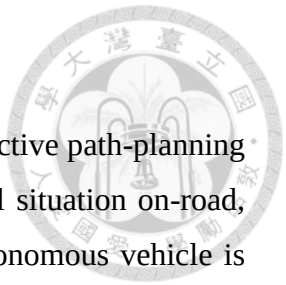
中文摘要

由於近年來無人載具的興起，對於自動駕駛技術的需求也越來越高。對於自駕車而言，傳統的路徑規劃演算法已經不能符合需要。因此，本論文提出一個即時、有效率的方法進行自駕車路徑規劃。為了符合現實狀況，我們納入自駕車對於環境的感知距離的限制。透過邊走邊探索的模式，自駕車在運行過程中不斷地更新地圖，並且在發現原先路徑不適用後，即重新進行路徑規劃。

本論文提出數個方法改進傳統的 A*演算法，使之適用於即時路徑規劃與避障。再者，透過修改啟發式函數與動態目標點的方法，處理滿足姿態要求的路徑規劃問題。最後，透過結合擴展卡爾曼濾波器(EKF)的方法，使線上路徑規劃適用於移動障礙物或時變地圖。

實驗與模擬結果顯示，這些修正有效降低傳統 A*演算法的計算時間，並且使其能夠根據需求的末端姿態做路徑規劃。模擬結果顯示，擴展卡爾曼濾波器確實可以提供足夠的障礙物預測資訊。再配合增加維度的路徑規劃演算法，我們可以實現平面運動的障礙物避障。

doi:10.6342/NTU201802627



Abstract

The design of on-road autonomous vehicles requires a real-time, effective path-planning algorithm for time-varying environment. To accommodate the real situation on-road, the restriction of the observation distance of the sensors on the autonomous vehicle is imposed. In this research, three methods are proposed to modify the offline A* path-planning algorithm for avoiding obstacles observed during path tracking. By redesigning the heuristic function and including the attitude requirements, we can obtain a suitable path adaptive to initial attitude and final attitude, as well as avoiding sharp turning which may cause difficulties in path-tracking. Moreover, Extended Kalman Filter(EKF) is integrated with the previous online path-planning algorithm to deal with moving obstacle.sSimulations and Experiments show that the proposed path-planning algorithm can reduce computation time significantly, and EKF can provide adequate information for moving obstacles such that the real-time path-planning and obstacle-avoidance are made possible.

Keywords: online path-planning、realtime path-planning、trajectory prediction、kalman filter

doi:10.6342/NTU201802627

目錄

doi:10.6342/NTU201802627

doi:10.6342/NTU201802627

圖目錄

doi:10.6342/NTU201802627

doi:10.6342/NTU201802627

doi:10.6342/NTU201802627

表目錄

doi:10.6342/NTU201802627

第1章 緒論

1.1 前言與研究動機

在利用動態規劃方法的路徑規劃演算法被荷蘭計算科學家 Edsger WybeDijkstra 提出之後，許多電腦工程師相繼投入關於利用計算機協助進行路徑規劃的研究。不管是如 A*那般透過修正 Dijkstra's Algorithm 使之維持最佳路徑但是具備更快速的探索效率，或者是如 RRT 那般另闢蹊徑利用隨機探索樹增加演算法的隨機性，都使得這個領域更加蓬勃發展。這幾年，許多機器學習與類機器學習的演算法也相繼投入解決路徑規劃問題的行列中。[1][2]

這幾年，許多需要自主運行的載具包含無人車、無人飛機、無人潛艦與機器人等蓬勃發展。對於載具而言，需要自主運行的條件主要分為三個部分，其一是能夠感知環境，其二是能夠根據任務規畫路徑與行為，其三是能夠精準地控制機構。無人車的關鍵在於各種感測器與系統的高度整合[3]。然而傳統的路徑規劃演算法面對高度動態與未知的環境時往往顯得捉襟見肘。傳統的路徑規劃演算法被應用於規劃行軍路線，或者規劃交通或旅遊路線等等，一般並不適用於動態環境或者地圖資訊不足的路徑規劃上。然而，在面對許多自主運行載具的需要時，重新理解並設計路徑規劃演算法變得至關重要。

本論文透過仔細檢視傳統路徑規劃演算法，並且根據不同的任務需求加以修改，使之可以更加適應現代無人載具在路上運行時的動態與未知環境。

doi:10.6342/NTU201802627

1.2 文獻回顧

1.2.1 路徑規劃演算法

路徑規劃演算法可透過圖論(Graph Theorem)和動態規劃(Dynamic Programming)的方法找尋從起點到目標點的最短路徑，然後自走車可以經由這條路徑到達目的地。在實務上使用路徑規劃演算法的時候，我們往往需要面對障礙物的避障問題。例如，當我們規劃一條路徑沿國道一號由台北開往高雄時，在台中遇到車禍堵住去路的時候，我們必須調整路徑才能到達目的地。路徑規劃演算法在面對障礙物的時候可以分為兩種處理方法。其中一種是設計一種適應避開障礙物的修正路徑方法用以閃開障礙物並走回原先預定的路徑上。另一種是先找出一條最佳路徑，然後再於遇到障礙物的時候重新進行路徑規劃，將避障與路徑規劃合於單一的演算法當中。本研究將使用後者的方法設計路徑，將路徑規劃演算法分為離線路徑規劃(Offline Path planning)與線上路徑規劃(Online Path Planning)，並結合兩者優勢用於自走車規劃適合其運動特性並能有效達到目的地的路徑。

在自走車運行之前為離線狀態(Offline state)，而開始運行之後為線上狀態(Online state)。而兩者的差別為對地圖資訊掌握的程度不同及路徑規畫目標與要求不同。若在離線狀態的時候，我們對地圖資訊全部已知，而且地圖上所有物件除了本身自駕車外全部都是靜態的，那我們就可以利用現已發展成熟的路徑規劃演算法，計算出最佳路徑並不再需要修正或者重新規劃。但在現實生活中面對的狀況，往往是在離線狀態時掌握的非常有限地圖資訊及隨時間變化的地圖。因此，我們除了利用最初掌握的資訊做離線路徑規劃(Offline Path Planning)外，還需要透過感測器得到有限探測距離的詳細地圖資訊，並依此做修正或者重新規畫路徑，才可以完成避障並到達目的地。我們稱這個重新規畫路徑的過程為線上路徑規劃(Online Path Planning)。

戴科斯徹演算法(Dijkstra's algorithm)是一種基於動態規劃(Dynamic Programming)方法發展而來的演算法[4]。其可以計算出非負邊(Non-negitive edge)

doi:10.6342/NTU201802627

權重圖的所有點到某個特定起始點的最短距離及路徑。由於戴科斯徹演算法是一種廣度優先搜索法(Breadth-First-Search)，它可以保證得到的路徑為最佳路徑，但卻比較沒有效率。於是，A*演算法(A* algorithm)被提出來改善戴科斯徹演算法的效率問題。它利用啟發函數(heuristic function)限制搜索範圍[5]，進而達到快速計算出一條滿意但不保證最佳的路徑。

上述的演算法皆適用於已知所有地圖資訊而且地圖不會隨時間改變的情況。在環境複雜化後，人們開始對地圖資訊改變時的路徑規劃演算法展開研究，並且提出了 LPA*演算法[6]。在地圖資訊些微調整時，LPA*演算法相對於 A*演算法只需要少量的更新與計算就可以得到新的路徑。這代表著當有障礙物被自走車偵測到的時候，LPA*演算法可以透過自走車當前的位置重新替自走車規劃一條可以避開前方障礙物的路徑。

1.2.2 卡爾曼濾波器

預測性濾波器(Predictive filter)在現代生活中經常被使用到，我們可以在自動化領域、電腦視覺、電腦圖像、軌道預測等領域見到他們的存在。這些基於貝氏定理(Bayesian rule)的濾波器將預測行為、測量與誤差融合在一個模型當中。[7]

卡爾曼濾波器於 1961 年由 kalman and Bucy 發表[8]，用於處理阿波羅軌道預測。透過模型預測與誤差修正兩步驟的演算，卡爾曼濾波器可以修正觀測量誤差並且動態的預測狀態。卡爾曼濾波器在實行預測步驟的時候假設系統為線性系統。然而，在實際情況往往會遇到許多非線性問題。因此，能夠處理非線性系統的卡爾曼濾波器在 NASA 的工作中被提出並使用[9]，稱為擴展卡爾曼濾波器(Extended Kalman Filter)。而後，一種使用蒙地卡羅方法(Monte Carlo method)的預測濾波器被提出，稱為粒子濾波器(particle filter)[10]。粒子濾波器幾乎為最彈性且適應性最高的預測濾波器。它將問題離散化為粒子，在取得足夠大量的粒子下幾乎可以處理任何非線性的問題。雖然粒子濾波器的適應性很廣，但其需要大量資料且需消耗大量計算過程的特性使其在應用上受到限制。往往在許多資訊較不足或者需要即時計算的問題當中，卡爾曼濾波器即使無法得到如粒子濾波器精確的結果，但仍較粒子濾波器更為適用。

doi:10.6342/NTU201802627

1.3 研究內容與成果

本研究提出探測範圍受到限制的路徑規劃演算法設計、姿態相關路徑規劃演算法設計、動態環境路徑規劃演算法設計。經由實驗驗證這些演算法設計的可行性，並且利用模擬比較與傳統 A*路徑規劃演算法的異同。

透過模擬與實驗可知，本論文提出的演算法較傳統 A*演算法在部分未知地圖與時變地圖的運行更有效率，並且可以符合初始姿態與末端姿態等姿態相關的路徑規畫要求。

1.4 論文架構

本章為緒論，接下來本文將依下列順序說明。

第二章介紹由動態規劃演變而來的路徑規劃演算法，包含 Dijkstra's algorithm 及 A*演算法。之後介紹利用結構修改、傳遞障礙物技巧、哈希表儲存節點資訊、姿態相關演算法設計等方法修改演算法以適應不同任務。

第三章介紹硬體架構與各感測器原理，包括 M3006V 網路攝影機攝影機、馬達編碼器等，並介紹實驗所使用的控制系統與路徑追蹤方法。

第四章呈現本研究提出路徑規劃演算法的模擬與實驗結果。其中包含線上路徑規劃演算法、動態環境路徑規劃演算法、姿態相關路徑規劃演算法的模擬與線上路徑規劃演算法、姿態相關路徑規劃演算法的實驗。

第五章為結論與未來展望。

doi:10.6342/NTU201802627

第2章　無人車路徑規劃演算法設計

本章介紹利用動態規劃方法設計最佳化路徑的 Dijkstra 演算法，並且利用啟發式函數(Heuristic function) 限制演算法的搜索範圍使其更能應付大範圍問題。之後提出離線與線上路徑規劃的任務與演算法需求，並依照兩方面修正路徑規劃演算法滿足線上路徑規劃的條件，最後透過修正啟發式函數 加入姿態因子達到姿態可納入考慮的路徑規劃演算法。

2.1 動態規劃與路徑規劃演算法

動態規劃是分治算法的延伸。如果一個複雜的問題可以分割成許許多多可以簡單求解的問題，那在完成這些簡單的問題之後就可以合併其結果，解決原先的複雜問題。整個動態規劃的過程當中就是不斷的讀取資訊，計算資訊，儲存資訊，直到解決問題為止。

再利用動態規劃解決問題的時候我們可以分成以下三個步驟。第一個是根據問題定義最佳值函數(Optimal value function)用以解決遞迴的子問題。第二個是推導出符合最佳值函數的遞迴模式(recurrent formula)。第三個是確認母問題在最佳值函數下的最佳解為何。然而，並非所有問題皆可使用動態規劃來解決。若一個問題可以使用動態規劃解決，它必須滿足以下兩個條件。第一個，這個問題必須能被分解成許多小問題。第二個，這個問題的所有子問題的最佳值必須能夠透過得到比其更小的子問題的最佳值而得到。由此可知，動態規劃尤其適合解決多層次的選擇問題(Multi-stage Decision Problem)。而路徑規劃問題本身並非多層次選擇問題。然而，我們可以透過將地圖網格(grid)化，將路徑規劃問題轉換為多層次選擇問題，進而使用動態規劃方法解決路徑規劃問題。又或者，我們可以利用圖(Graph)、節點(node)、邊(edge)分別表示地圖、地標與距離將問題轉換為多層次選擇問題。

以下，我們將利用圖來描述一個路徑規劃問題，並且嘗試使用動態規劃的方法解決路徑規劃問題。

doi:10.6342/NTU201802627

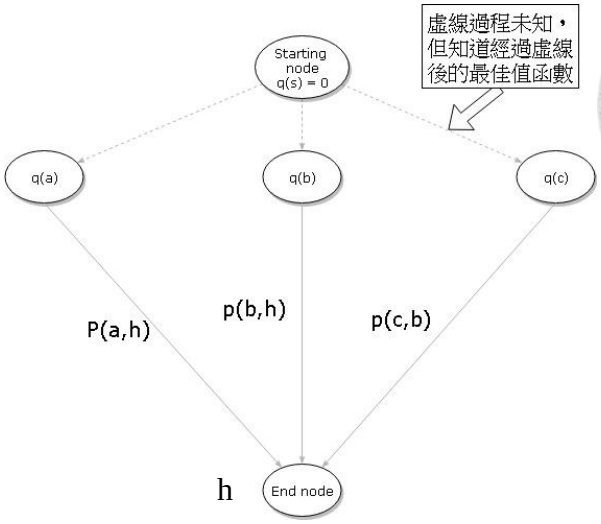


圖 2-1 態規劃方法解決路徑規劃問題示意圖

第一步：q(h)為最佳值函數表示從起點到 h 點所需要的路程，即所經過的邊長相加。

第二步：根據問題推導出符合最佳值函數的遞迴模式。

$$\mathrm{q(h)} = min\begin{cases} q(a) + P(a,h) \\ q(b) + P(b,h) \\ q(c) + P(c,h) \end{cases} \qquad (2.1)$$

第三步：由上述遞迴模式可以知道從起始點到 h 點的最短距離為上三式中的最小值。

2.2 路徑規劃演算法

2.2.1 戴科斯徹演算法

利用 2.1 節建構的路徑規劃問題的動態規劃模型，可以最終得到所有非負邊長圖(Non-negitive edge weight graph)的任何一點到起點的最短路徑。這個概念最早由荷蘭電腦科學家艾茲赫爾·戴克斯徹在 1956 年提出，而稱為戴科斯徹演算法。以下為此演算法的虛擬碼。

doi:10.6342/NTU201802627

```
Algorithm DijkstraDistances (G, s)
    Q ← new heap-based priority queue
    for all v ∈ G.vertices()
        if v = s
            v.setDistance(0)
        else
            v.setDistance(∞)
        l ← Q.insert(v.getDistance(), v)
    v.setEntry(l)
while ¬Q.empty()
    l ← Q.removeMin()
    u ← l.getValue()
    for all e ∈ u.incidentEdges() { relax e }
        z ← e.opposite(u)
        r ← u.getDistance() + e.weight()
        if r < z.getDistance()
            z.setDistance(r)
            Q.replaceKey(z.getEntry(), r)
```

圖 2-2 Dijkstra's algorithm 虛擬碼[4]

此虛擬碼代表的意義如下：

首先，將所有點的到起始點的距離設為無限大，起始點到自己的距離設為零。再來，利用圖更新點到起始點的距離，其餘則不變。再來，將距離起始點最短的點設為中繼點，若經過它再到起始點比較短則取代原先的距離資料。之後在找僅次於此中繼點的點為中繼點，繼續更新資料。以此類推即可得到所有的點到起始點的最短距離。

在完整提供地圖資訊的情況下，戴科斯徹演算法可以保證提供任何一點到起始點(source)的最短距離。戴克斯徹演算法為先解決並儲存被分解(Decomposition)到較低階和簡單的最短距離問題，然後透過遞迴將這些子問題的解用在解決母問題上。

doi:10.6342/NTU201802627

然而，若考慮到計算速度問題，戴科斯徹演算法的時間複雜度如下所示

若一張圖具有 V 個節點(vertex)和 E 條邊(edge)，其時間複雜度為：[4]

$$O(V^2) \tag{2.2}$$

若使用費波納契堆(Fibonacci Heap)和相鄰表(adjacency list)優化則可以將時間複雜度優化成：[4]

$$O(E + VlogV) \tag{2.3}$$

2.2.2 A*演算法

Dijkstra 演算法的計算量會隨著節點數的增加而大幅增加。然而，大多路徑規劃問題的地圖很大，節點數也很多，但是有很多地區其實很明顯的知道大可不必搜索。為了改善這個問題，A*演算法使用啟發函數(heuristic function)來指導並限制 Dijkstra 演算法的搜索方向與範圍。傳統的 A*演算法使用起始點與終點的距離來當作啟發式函數。並且以此來限制 A*演算法的搜索，使之比起 Dijkstra 演算法更有效率。以下為 A*演算法的虛擬碼：

開表(open list): 包含所有已經被探索過但還沒有計算最佳值的節點。

關閉表(close list): 包含所有已經被探索且已經計算出最佳值的節點。

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17    }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

圖 2-3 A* algorithm 虛擬碼[12]

doi:10.6342/NTU201802627

此虛擬碼的開表中的點是否放入關閉表並且成為下一個搜索的原點需要參考其啟發式函數。在一個關閉表中，只有啟發式函數與 g(n)的合值最小的點能被放入關閉列並執行下階段的搜索。

2.2.3 啟發式函數的介紹

A*演算法不同於 Dijkstra 演算法，它並非一個完整的演算法[12]。在不同的啟發式函數作用下，A*搜索演算法的行為也會有所不同。首先，A*的最佳值函數如下所示：

$$f(n) = h(n) + g(n) \tag{2.4}$$

若 h(n)為零，則 A*演算法就會轉變為 Dijkstra 演算法。若在整個計算過程當中，h(n)都比 g(n)小，那 A*演算法也會向 Dijkstra 一樣保證提供最短路徑。而當 h(n)的影響越大，A*演算法的搜索速度就越快，範圍就越小但是也越有可能錯失最佳路徑。最後，當 h(n)遠大於 g(n)的時候，A*演算法轉變為貪婪演算法(Greedy Search)[13]。

由於我們的地圖將作網格化，在此介紹經常被使用的網格地圖啟發式函數。

1. 曼哈頓啟發式距離(Manhattan distance)

$$h(n) = \alpha * (|n.x - goal.x| + |n.y - goal.y|) \tag{2.5}$$

其中，α為啟發式因子(heuristic factor)，若希望 A*演算法越快，越接近貪婪演算法(Greedy Search)則提高其值。若希望搜索越全面，越接近 Dijkstra's 演算法則降低其值。然而，在本論文之後增加 h(n)的功能之後，啟發式因子會擴展為兩項，分別代表不同功能的比例。

2. 尤拉啟發式距離(Euclidean distance)

$$h(n) = \alpha * \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2} \tag{2.6}$$

由上式可以簡單看出，尤拉啟發式距離就是節點和終點的真實距離。

3. 正交啟發式距離(Diagonal distance)

$$h(n) = \alpha * \left[(|n.x - goal.x| + |n.y - goal.y|) + \left(\sqrt{2} - 2\right) * \min(|n.x - goal.x|, |n.y - goal.y|)\right] \tag{2.7}$$

doi:10.6342/NTU201802627

相較於曼哈頓啟發式距離只是 x 和 y 的座標相差的合，正交啟發式距離列入了斜向步距，是介於只計算網格距離的曼哈頓啟發式距離和完全脫離網格計算真實距離的尤拉啟發式距離中間。

下圖為三種啟發式距離的示意圖。綠色線條為曼哈頓啟發式距離，藍色為正交啟發式距離，紅色為尤拉啟發式距離。A*演算法與 Dijkstra 演算法的差別就在於啟發式函數。啟發式函數可以避免搜索時漫無目的地搜索，像個指導者一樣給予搜索演算法指引。另外，經過設計之後，啟發式函數還可以有其它功能，這部分將在路徑規劃演算法處理姿態限制部分進行說明。

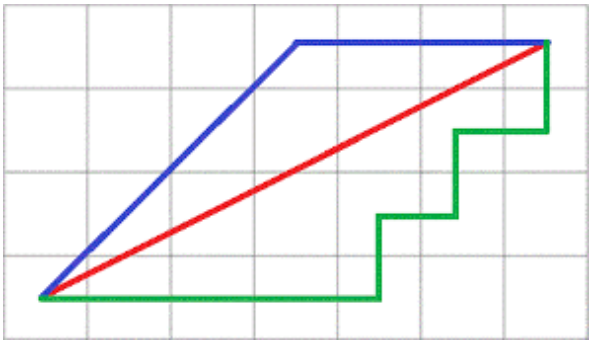


圖 2-4 不同啟發式函數的比較圖

2.3 線上路徑規劃

線上路徑規劃演算法的目標是處理在初始地圖資訊不全情況下的路徑規劃問題。實務上，載具的探測距離是有限制的。探測到新增的障礙物以後，原先的路徑可能就不再適用。我們必須先更新地圖資訊之後再進行路徑規劃，然而，由於這個規劃是在載具運行中的時候進行，對於效率的要求就更高了。

2.3.1 線上路徑規劃的架構與用途

路徑規劃演算法在面對部分已知地圖(Partially known map)可以分成兩個部分。第一個部分為離線路徑規劃。先利用初始已知的地圖資訊與 A*演算法將路徑進行初步規劃。然後，載具即根據這個初始的路徑運行。當載具探測到這個初始路徑上有障礙物的時候，即進行第二部分，也就是線上路徑規劃。線上路徑規劃就是根據

doi:10.6342/NTU201802627

新探測的障礙物更新原先規畫路徑時的地圖，然後再以現在的位置為初始點重新進行路徑規劃。以下為此演算法的流程圖：

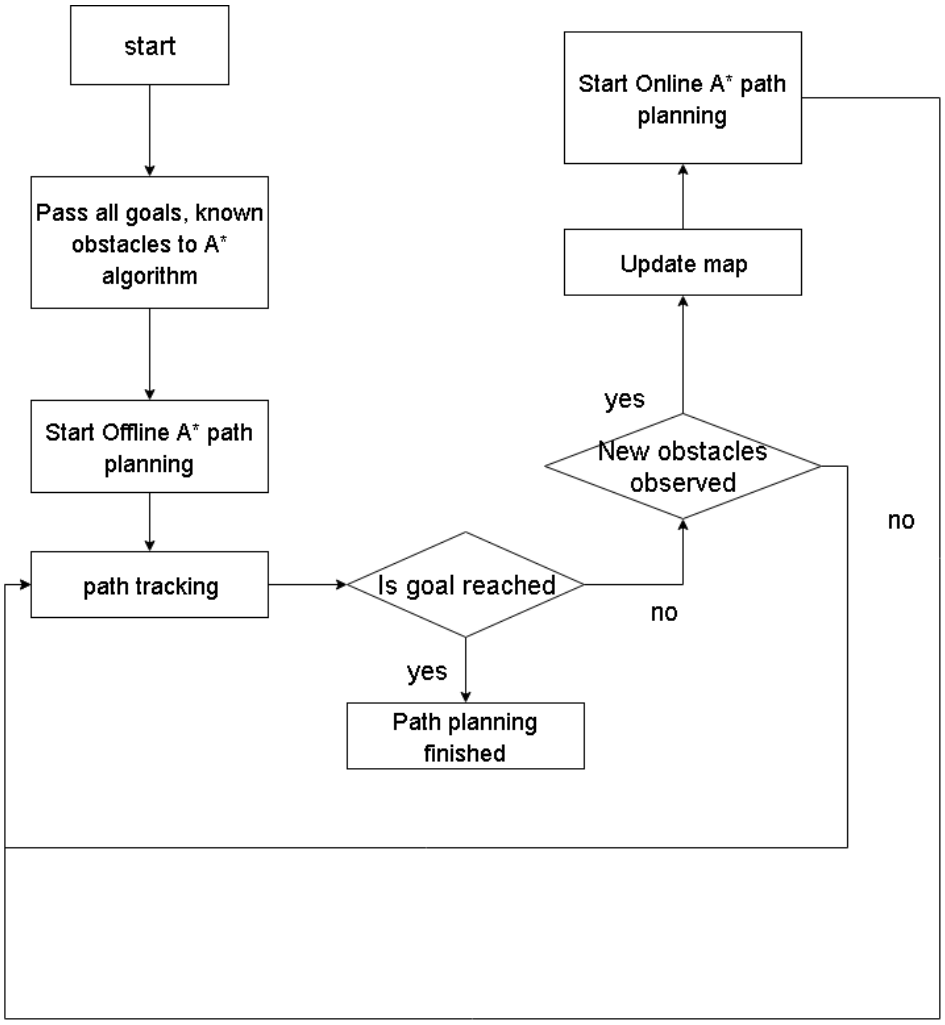


圖 2-5 部份已知環境的路徑規劃演算法流程圖

doi:10.6342/NTU201802627

2.3.2 適應線上路徑規劃的 A*演算法

原先的 A*演算法本身已經可以適用線上路徑規劃。但是，在面對一些地圖資訊的小修改時，A*演算法往往需要大幅修改路徑與運算才能再次得到適用的路徑。這可能導致使用 A*演算法在線上路徑規劃的時候停頓太久。面對一次修改一小部分的線上路徑規劃，我們也許可以找到一個演算法只需要使用小幅度的運算即可得到新路徑。在觀察 A*演算法實作的時候，我們發現有兩個儲存節點(Vertex)的資料列，分別為開放列表(Open list)和關閉列表(Closed list)。然而，根據演算法的設計原則，我們可以增加儲存的資訊以減少消耗的時間，我們透過增加 A*容器的數量，儲存曾經被修改或者永遠不可能再修改的節點資料列。以下為比較表：

表 2-1 資料表比較表

Traditional A* algorithm	structure modified A* algorithm
Open list Closed list	Open list Closed list Dead list Raised list

而在 A*演算法當中我們可以將其分成兩個部份，一個是擴展範圍，另一個是檢查節點資訊是否改變。首先，當在作路徑追蹤時，若遇到原先路徑上有障礙物使的原先路徑失效時啟用此演算法。首先，在該障礙物點存取其背向點(Back Point)，之後檢視其是否為非障礙物點，若不是則繼續找背向點，若是則以該點為中心開始擴展。首先，找尋該中心點的所有臨點(neighbor)，將符合周圍除了背向點之外所有最佳值均大於它的鄰點放入死亡點表(Dead List)如同障礙物永不再搜索。之後，將剩下的點使用背向點作檢查，若其背向點的點為提升點(raised list)，或者其背向點的最佳值比自己的最佳值高，則將此點納入提升點當中。之後，再利用其鄰點更新旗標與背向點，最後留下最小值，則此點更新完成，繼續利用同 A*的方法搜索。以此迴圈不斷搜索，直到找到現在的自駕車位置為止。

doi:10.6342/NTU201802627

而除了將原先的 A*演算法擴展為兩個步驟並增加容器儲存資料外，計算最佳值函數的方法也有所改變。以下比較表比較 Dijkstra's 演算法，A*演算法和適應線上路徑規劃的 A*演算法計算最佳值函數的方法

表 2-2 最佳值函數與 key 比較表

	Dijkstra's Algorithm	A* Algorithm	structure modified A* algorithm
最佳值函數	g(n)	g(n) + h(n)	g(n) + h(n)
flag	none	none	$g(n) + \min(h(n)_{new}, h(n)_{old})$

由前一段可以得知，旗標的作用在於當發現其背向點的最佳值被提高時，用來比較新背向點與舊背向點何者更適合並最後留下最佳者更新最佳值。

2.3.3 哈希表(hash_map)優化節點資料儲存

哈希表為根據鍵(key)和表(map)透過某個映射函數互相對應。透過計算映射函數尋找，修改，新增哈希表中的資料均可以維持在常數時間複雜度 O(1)。[14]

在離線路徑規劃的時候，因為沒有上升表和下降表，基本上確定的最佳值不會被修改，因此再次存取節點和節點資料的需求相對降低。我們可以利用簡單的優先序列(priority queue)解決資料儲存問題。然而，在線上路徑規劃的時候，我們需要重複使用不需被修改的節點資訊，而且快速修改需要被修改的節點資訊，因此用完即丟而且存取速度皆不符合需求。以下為比較哈希表和優先序列的時間複雜度表。

表 2-3 Hash Table 與 Priority Queue 比較表

	Hash Table(with out collision)	Priority Queue(Binary tree)
insert	O(1)	O(log n)
delete	O(1)	O(log n)~O(1)
modify	O(1)	O(log n)

doi:10.6342/NTU201802627

因此在線上路徑規劃的時候引入哈希表可以明顯加快其在對應環境變化時的運算。我們一般稱鍵和表之間對應的函數為哈希函數(hash function)。以下為哈希表運作的示意圖：

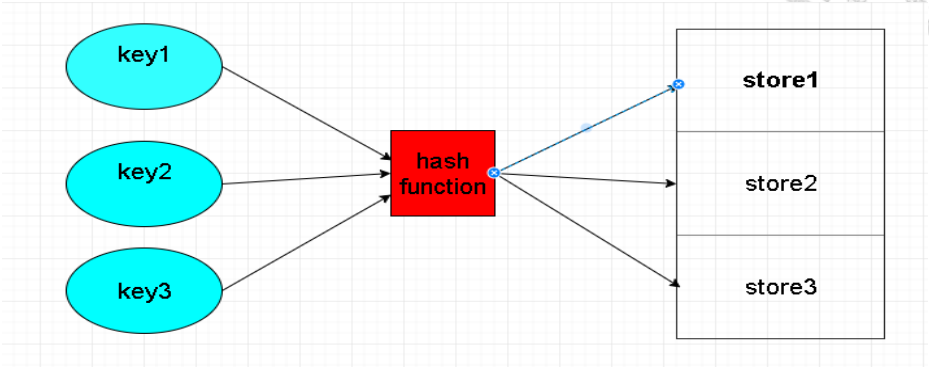


圖 2-6 哈希表示意圖

由圖可知，哈希函數(hash function)的設計對於其運作有至關重要的影響。有可能兩個不同的鍵(key)卻對應到表中的同一個位置。如果有這個情況發生，我們稱之為對衝(Collision)。若有對衝發生，則時間複雜度將不再是常數，最慘可能變為線性時間複雜度。對於處理哈希表的碰撞問題，我們可以將碰撞的值存成一個鏈結表(Linked List)再逐一檢視篩選。然而，最好的方法其實是盡量分散每個鍵的對應位置並盡量避免碰撞發生。以下為碰撞發生時的示意圖：

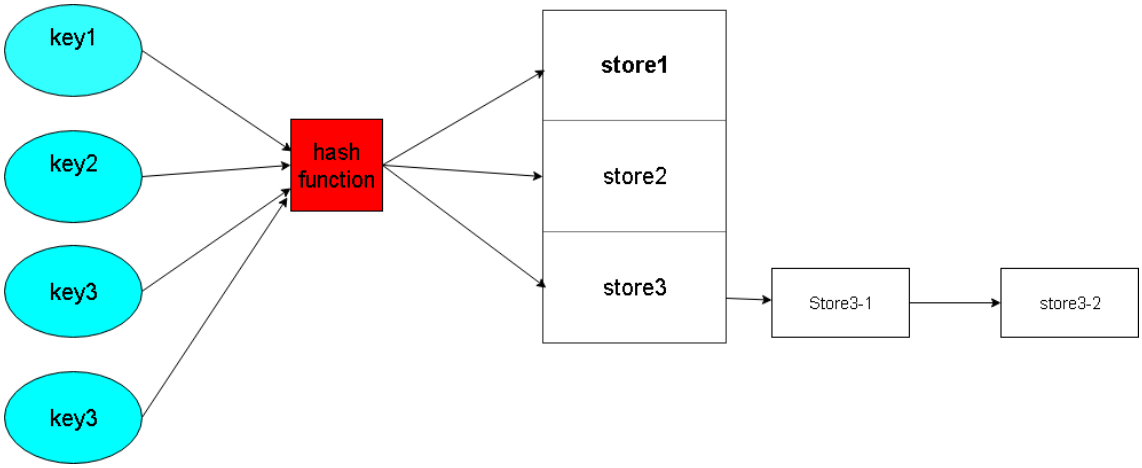


圖 2-7 哈希表的碰撞(collision)示意圖

以下為簡單的哈希表與對衝的例子。若有一個 key:x 儲存資料為 y(x) ，則可以設計以下哈希函數：

$$h(x) = x^2 + 1 \qquad (2.8)$$

doi:10.6342/NTU201802627

則可以知道，如果我們知道鍵 x，則可以透過存取哈希表中 h(x)位置中的資料即可找到或者存取修改資訊。然而，這個方法只需要寄算 h(x)，速度與 x 的大小無關，因此時間複雜度為常數。然而，我們可以發現在這個哈希函數中，鍵 x 和鍵-x 均會映射到同一個哈希表中的位置，我們就稱 x 和-x 對衝。

然而，面對路徑規劃問題的時候，由於我們將地圖網格化，我們可以透過非常簡單的哈希函數設計簡單地避開所有碰撞的可能。如果每個節點都有其特殊的哈希值，則這個哈希表不可能產生對衝。如果一個二維的節點地圖其 x 軸和 y 軸的大小均為 L，則我們的哈希函數如以下方法設計。

$$\text{hashf}(x, y) = x + (L + 1) * y \tag{2.9}$$

然而，若此地圖的 x 軸和 y 軸的長度不等，分別為 L 和 M，則哈希函數設計如下：

$$\text{hashf}(x, y) = x + \max(L, M) * y \tag{2.10}$$

由於當每個鍵均可在哈希函數中產生唯一的哈希值而儲存到唯一的位置，並不會跟其它 key 衝突，因而不可能產生對衝的情形。 經過這樣的設計我們可以達到避免節點和節點之間碰撞的可能性，也省去複雜的碰撞處理流程。這樣的修正對於節點資訊的儲存和處理的速度均有明顯的改善，第四章的實驗結果可以知道，儲存資料方式的修改奇益處不亞於修改演算法本身。

doi:10.6342/NTU201802627

2.3.4 姿態相關路徑規劃演算法設計

在 2.2.3 節的討論中我們可以發現，啟發式函數對於 A*的搜索和最佳值均有影響。此外，我們也可以透過修改啟發式函數而控制路徑搜索以達到特定目的。而在這裡我們希望透過修改啟發式函數而達到姿態相關的目的。

2.3.4.1 初始姿態相關路徑規劃與旋轉限制路徑規劃

進行控制器測試的時候，我們發現一件會影響實驗結果的現象。在動態路徑規劃的過程當中，我們經常需要從某個車子的所在位置開始做重新的路徑設計。然而，傳統的 A*演算法並沒有計入姿態。因此，如果在路徑一開始就要求載具做大幅度的轉彎會造成其追線上的困難。以下為大幅度旋轉的示意圖：

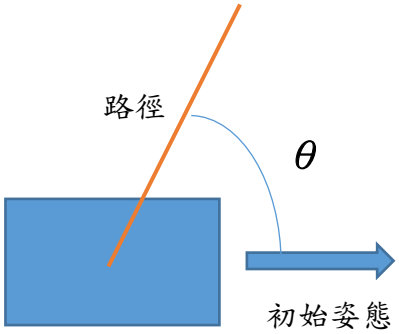


圖 2-8 初始大幅度旋轉是意圖

另外，在初始路徑規劃的時候也經常遇到這樣的問題。A*在一開始規畫路徑的時候並沒有考慮初始的車輛姿態，而是根據啟發函數的指導往目標的方向規畫路徑。因此如果可以讓演算法根據不同的初始姿態而做不同的路徑設計，則可以減少車輛控制在追蹤路徑的時候的誤差。舉例來說，若有一個地圖沿對角線對稱，而起點和終點也都在對角線上，則我們應該根據初始姿態的不同而往右上或者往左下進行路徑規劃。如果車子的初始姿態指向偏左上的方向，則我們應該沿左上規畫路徑，反之則往左下。我們透過重新設計啟發式函數來達到這個兩個作用

(1) 在離線和線上路徑規劃時，重新規劃納入姿態因子以避免規劃出難以追蹤的路徑

(2) 在遠離初始點的時候逐步減少姿態因子對啟發式函數的影響，以避免

doi:10.6342/NTU201802627

造成路徑遠離最佳路徑

以下為路徑相關啟發式函數的設計，以滿足以上兩個目的：

$$h_{ati}(n) = \alpha * h(n) + \beta * \cos(\Delta\theta)/search\_order \quad (2.11)$$

其中，α為啟發式因子(heuristic factor，β為 attitude factor，h(n)為 A*演算法使用的啟發式函數，Δθ為載具初始姿態和路徑點和初始點形成的向量之間所夾的角度差，search_order為路徑上的點的次序，起使為 1。論文中的所有路徑相關演算法所使用的α和β分別為 1 和 200。而所使用的地圖大小為 1000x1000。而 search_order 的作用則是使的路徑影響在離初始點越遠的時候逐漸降低。

2.3.4.2 末姿態相關路徑規劃演算法

若我們希望載具在到達終點時依照某個特定姿態行至終點，則傳統的 A*演算法亦無法達到目的。然而，在生活中許許多多的應用都很要求特定的末姿態，例如從地下停車場入口規畫路徑進入停車格。若無法提供正確的末姿態路徑，則會造成最後必須重新調整路徑等等影響效率的事情發生。因此，在此節將提出一個方法解決末端態決定問題，並且將其融入路徑規劃演算法當中。以下為這個方法的示意圖：

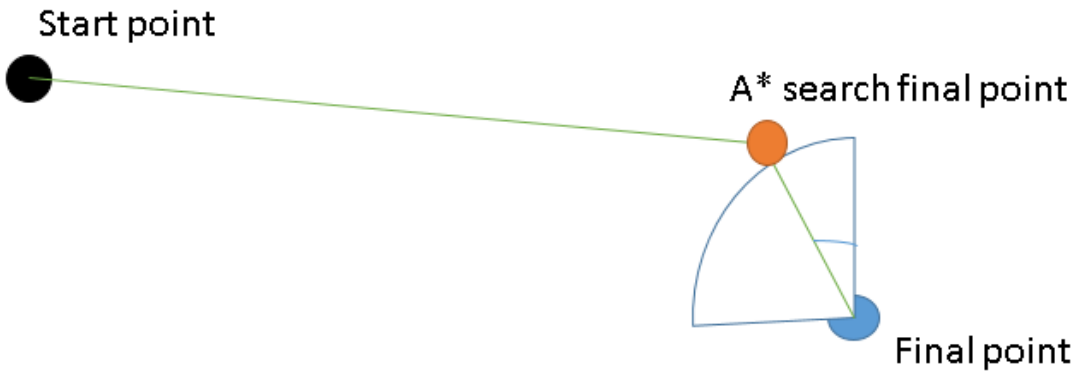


圖 2-9 動態搜索目標示意圖

首先，先確認終點附近無障礙物地域的範圍，並且選擇一個半徑且這個半徑必須小於無障礙物地域範圍的半徑。接著，依照需要的末端姿態在以終點為圓心且以此半徑畫出的圓找一個點使的這個點到終點的向量為末端姿態。A*搜索實際上以這個新的搜索終點為終點，而規畫出的路徑最後再加上 A*搜索終點與終點的連線，這樣即可以保證載具在追完這個路徑之後形成指定的姿態。

doi:10.6342/NTU201802627

2.4 動態環境路徑規劃

一般的路徑規畫演算法提供靜態環境的路徑規劃，也就是若環境中的障礙物資訊不變，則透過前幾節所述的方法我們可以得到最佳或者堪用的路徑。然而，若障礙物在環境中隨時間改變，則問題將變得複雜許多。當得知障礙物 A 的位置時，透過路徑規畫演算法計算出一個可以避開 A 的路徑。但當開始追蹤路徑時，A 卻已不在該位置且可能出現在的圖的其他地方造成載具路徑的失效。面對這樣的問題，本論文提出使用擴展卡爾曼濾波器預測障礙物與時間的關係，並且使用適應時變環境的改良式 A*演算法進行動態環境的路徑規劃。最後，提出結合動態與靜態避障路徑規劃。

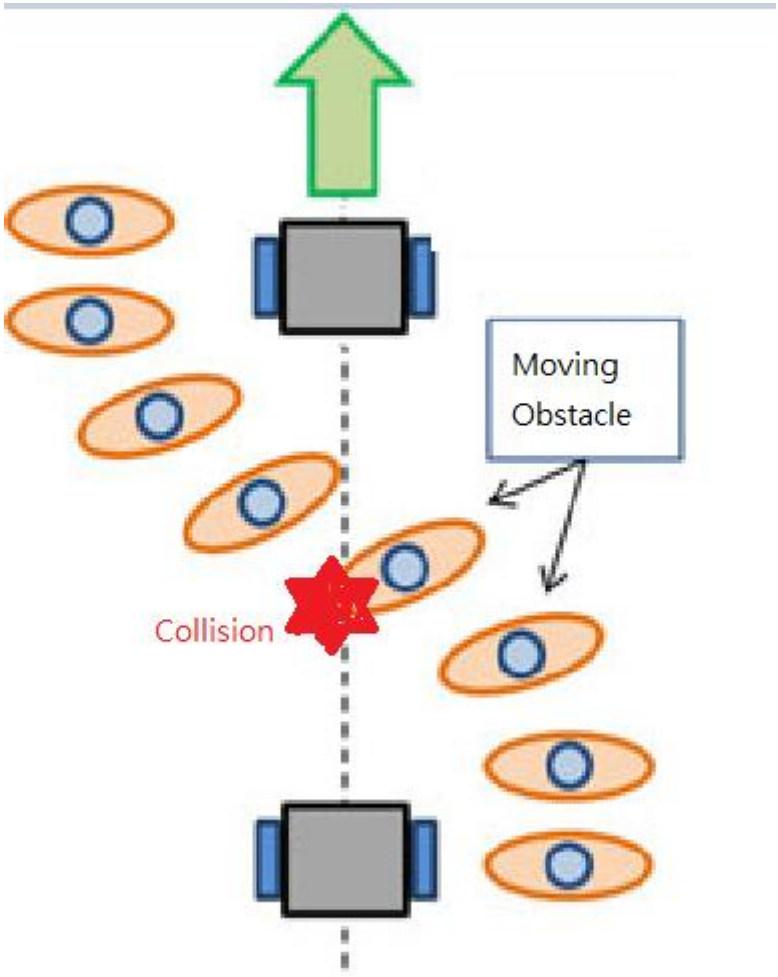


圖 2-10 動態環境避障示意圖

doi:10.6342/NTU201802627

2.4.1 動態環境路徑規劃方法

在面對動態環境的路徑規劃時，可以將演算法分為兩個部分。第一個為觀測障礙物行為並預測障礙物行為。第二個為根據所得的資訊實行路徑規劃。以下為動態路徑規劃方法的流程圖：

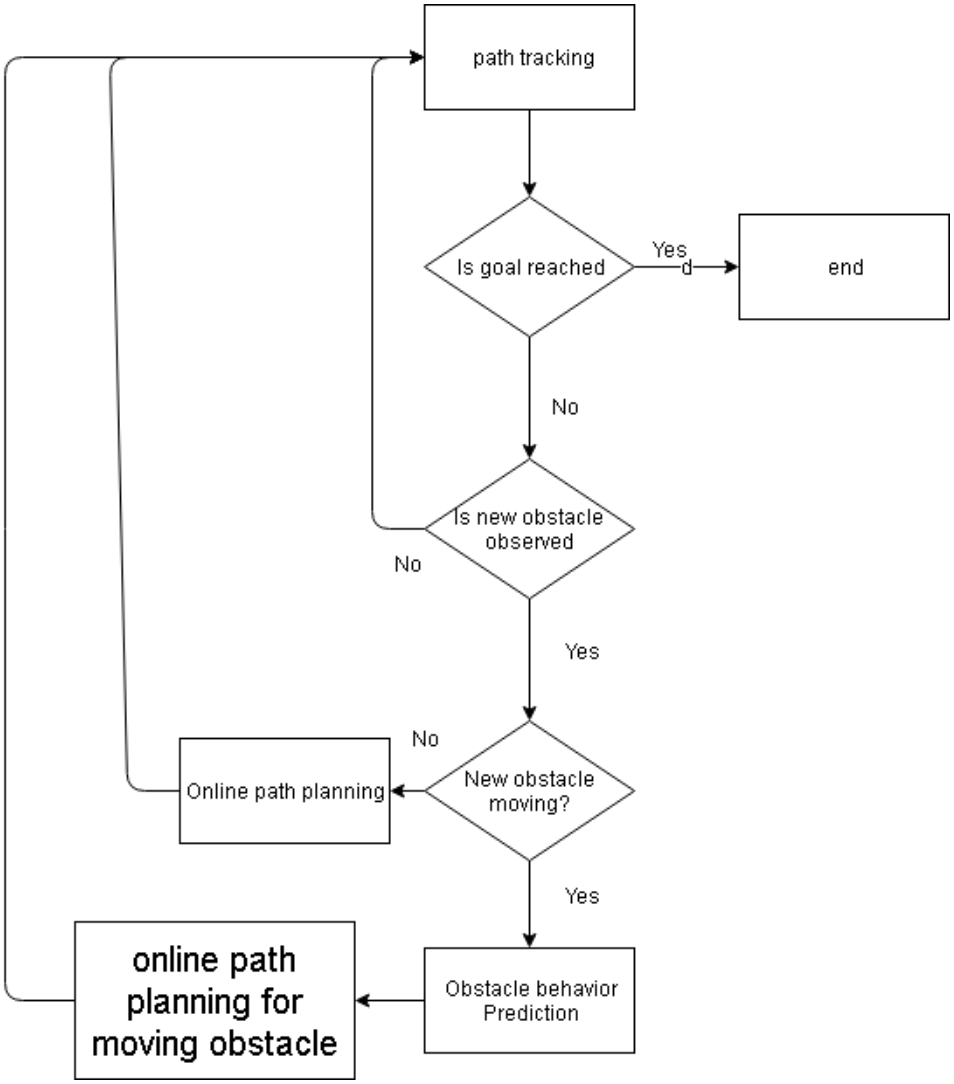


圖 2-11 動態環境路徑規劃示意圖

其中，障礙物預測的部分不一定是真的去預測障礙物，也包含根據障礙物資訊做出不同的對策。例如，根據觀測的障礙物速度不同擴增障礙物的大小，以達到避開動態障礙物的目的。[15]

本論文使用的方法則是利用預測性濾波器去預測障礙物的位置與時間關係，並且將預測結果提供給路徑規劃演算法進行路徑規劃以達避開動態障礙物的目的。

doi:10.6342/NTU201802627

以下將逐步介紹這些用來避開動態障礙物的工具，包含擴展卡爾曼濾波器和適應時變環境的路徑規劃演算法。

2.4.2 擴展卡爾曼濾波器

2.4.2.1 預測性濾波器介紹[16]

在許多應用科學當中，我們使用數學模型去描述與處理現實問題。一個合適的數學模型讓我們更容易推斷實體的狀態。然而，我們可以找到一組狀態參數並透過數學模型來描述實體的狀態。這些狀態參數未必是唯一，我們可以針對我們的需求或者系統的特性找到適合的狀態參數。我們稱這個會隨著時間改變描述狀態參數的數學模型為系統(system)。

有時候，當系統當中的某個參數無法或者難以被測量的時候，我們可以透過量測其它與這個參數相關的物理量並且透過這些量測值來描述這個參數。而這些間接的量測我們可以稱為測量值(observation)，並且可以用來推斷系統中難以被測量的參數。

預測性濾波器可以透過以下兩個步驟為系統估測出一個最接近現實的狀態。首先，利用系統動態的數學模型預測系統的下個狀態和不確定性(uncertainty)。接著，利用測量得到的測量值去修正前一個預測所得到的狀態。之後不斷往復前兩個步驟並不斷的為系統進行預測與更新。卡爾曼濾波器(Kalman Filter)為最簡單且最常被使用的預測性濾波器。它利用高斯隨機變數(Guassian random variable)描述系統的不確動性(Uncertainty)，並且利用線性的動態模型與觀測值描述系統。

由於傳統的卡爾曼濾波器假設系統的不確定性為高斯分布，而且利用線性動態模型描述系統，因此仍有許多問題無法利用卡爾曼濾波器進行預測與估計。例如：一個沿著拋物線自由落體運動的物體就無法透過傳統的卡爾曼濾波器預測並估計其位置。本研究使用的系統動態模型也屬於非線性模型，因此亦無法使用傳統的卡爾曼濾波器進行預測與估計。因此，有許多適應更多狀況的預測性濾波器也逐漸被提出併使用。在研究中，我使用擴展卡爾曼濾波器(extended Kalman Filter)

doi:10.6342/NTU201802627

對偵測到的障礙物位置進行預測與估計，並將預測出來的位置提供給路徑規劃演算法進行避障。

2.4.2.2 系統簡介

預測性濾波器(Predictive Filter)依靠數學模型描述系統。在這節中，我們將簡述我如何設計面對路徑預測問題的數學模型。

若我們使用 n 個參數描述系統狀態，則我們可以將系統狀態描述成一個 n 維的向量如下所示；

$$\overrightarrow{x_t} \in R^n \tag{2.12}$$

而在預測障礙物路徑的過程中，我們關心下個狀態時的自走車位置與姿態。然而，若我們要預測下個狀態時的障礙物位置，我們也需要使用系統當下的速度去建立前一個狀態與下一個狀態之間的關係。因此，我們使用位置、姿態與速度為系統的狀態參數。

2.4.2.3 卡爾曼濾波器

若系統為線性，高斯隨機變數很適合拿來表示系統狀態。我們可以用以下算式表示線性系統：

$$\overrightarrow{x_k} = A_{k-1}\overrightarrow{x_{k-1}} + \overrightarrow{w_{k-1}} \tag{2.13}$$

$$\overrightarrow{y_k} = H_k\overrightarrow{x_k} + \overrightarrow{v_k} \tag{2.14}$$

其中：$\overrightarrow{x_k}$：n 維度向量表示系統在 t=k 時的狀態

$\overrightarrow{y_k}$：m 維度向量表示系統在 t=k 時的觀測量

A_{k-1}：nxn 轉移矩陣，描述系統狀態 k 與 k-1 的動態關係

$\overrightarrow{w_{k-1}}$：n 維度預測雜訊

$\overrightarrow{v_k}$：m 維度觀測雜訊

對於每個迴圈，卡爾曼濾波器總共會執行兩個步驟，第一個為預測(prediction)，第二個為修正(correction)。下圖為預測與修正步驟的介紹：

doi:10.6342/NTU201802627

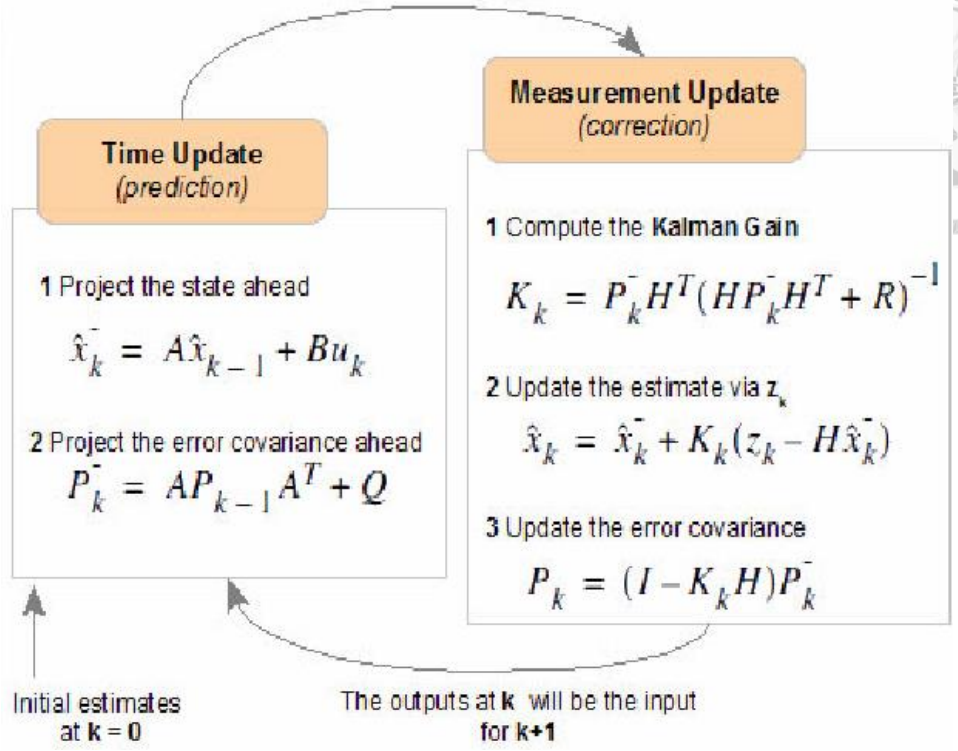


圖 2-12 卡爾曼濾波器流程示意圖 [13]

其中，P_k為估計誤差協方差，Q 為狀態噪聲的協方差矩陣，滿足以下方程式：

$$w_k \sim N(0, Q_k) \tag{2.15}$$

其中，R_k為量測噪聲的協方差矩陣，其滿足以下方程式：

$$v_k \sim N(0, R_k) \tag{2.16}$$

由於我們面對的問題也就是障礙物位置的量測是用全域影像測量，其測量精度很高，因此，我們使用的量測噪聲的協方差矩陣為值很小的對角矩陣。而狀態噪聲的協方差矩陣則為相對較大的對角矩陣。以下為本論文所使用的卡爾曼濾波器系統：

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \\ v_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \\ v_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} v_{k-1} * \cos(\theta_{k-1}) * t \\ v_{k-1} * \sin(\theta_{k-1}) * t \\ \omega_{k-1} * t \\ 0 \\ 0 \end{bmatrix} + \overrightarrow{w_{k-1}} \tag{2.17}$$

doi:10.6342/NTU201802627

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} * 0.01 \quad (2.18)$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.19)$$

由上式可知，此系統並非線性系統，因此不符合卡爾曼濾波器所假設的系統動態為線性的限制。因此，我們必須改用以下所介紹的擴展卡爾曼濾波器。其利用泰勒展開等線性化的手段使的卡爾曼濾波器可以處理非線性問題。

2.4.2.4 擴展卡爾曼濾波器

$$\vec{x}_k = f(\vec{x}_{k-1}, \vec{\omega}_k) \quad (2.20)$$

$$\vec{y}_k = h(\vec{x}_k, \vec{v}_k) \quad (2.21)$$

擴展卡爾曼濾波器的狀態轉換和狀態觀測皆不需要是線性，而改用函數表示。然而，我們也無法如上小節直接利用 f 和 h 應用在協方差上，而是要改用 Jacobian 矩陣更新模型，如下所示：

$$P_k = F_k P_{k-1} F_k^T + Q_k \quad (2.22)$$

$$F_k = \frac{\partial f}{\partial x} | \vec{x}_{k-1} \quad (2.23)$$

$$H_k = \frac{\partial h}{\partial x} | \vec{x}_{k-1} \quad (2.24)$$

在這個架構下，我們就可以將原先非線性的系統線性化而使用卡爾曼濾波器上進行預測與修正。其預測與修正的流程如下所示：

doi:10.6342/NTU201802627

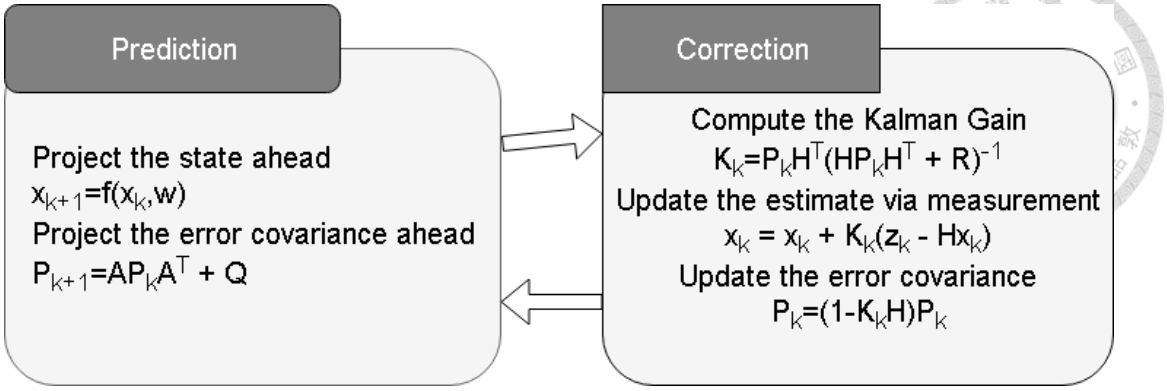


圖 2-13 擴展卡爾曼濾波器流程示意圖

2.4.3 適應時變環境的路徑規劃演算法

過去，當 A*演算法在擴展搜索點的時候，我們使用八方向搜索法。總共往以下八個向量搜索(1,0), (0,1), (1,1), (-1,0), (0, -1), (-1,-1), (-1,1), (1, -1)。而如果由 U 點可以搜索到 V 點，則也可以保證由 V 點也可搜索到 U 點。為了避免永無止境的重複搜索，所以 A*演算法有關閉表用以限制已經完成搜索的點被重新搜索。

然而，在時變環境下這些規則已經不適用。首先，我們定義時間為一個維度的物理量。所以，若對二維的時變環境做路徑規劃，實際上等於對三維(2D+t)地圖做路徑規劃。

$$x(t) \in R^n \approx x' \in R^{n+1} \tag{2.25}$$

透過擴展一個維度，我們可以把原先的時變的圖轉成靜態，而後路徑規劃演算法就可以適用這樣的問題。但是，我們可以明顯地知道，空間中的軸性質還有時間軸性質有所不同。空間上的位置可以倒退，但是時間不能倒退也不能停留。由此可知，本節第一段所提的那八個向量勢必有所修正。而由於"由 U 點可以搜索到 V 點，則也可以保證由 V 點也可搜索到 U 點"這個假設已經不符，所以我們必須把相鄰點(neighbor)分割為後點(successor)和前點(predecessor)。其中，後點是由 U 點可以前往的點，而前點則是可以前往 U 點的點。以下為示意圖：

doi:10.6342/NTU201802627

圖 2-14 successor 和 predecessor 示意圖

這個圖表示 U 為 V 的前點且 V 為 U 的後點。而下表為所有前點和後點的向量集合；

表 2-4 successor 與 predecessor 表

predecessor	successor
(0,0,-1)	(0,0,1)
(1,0,-1)	(1,0,1)
(-1,0,-1)	(-1,0,1)
(0,1,-1)	(0,1,1)
(1,1,-1)	(1,1,1)
(-1,1,-1)	(-1,1,1)
(0,-1,-1)	(0,-1,1)
(1,-1,-1)	(1,-1,1)
(-1,-1,-1)	(-1,-1,1)

任何不符合在此表列內的 predecessor 和 successor 都是不合法的，將會造成路徑規劃演算法的錯誤。另外，時域擴展地圖也有相關類似的問題，一個障礙物不可能從時域擴展地圖上的某個時間消失，或突然出現。下圖為時域擴展地圖是否合理的示意圖：

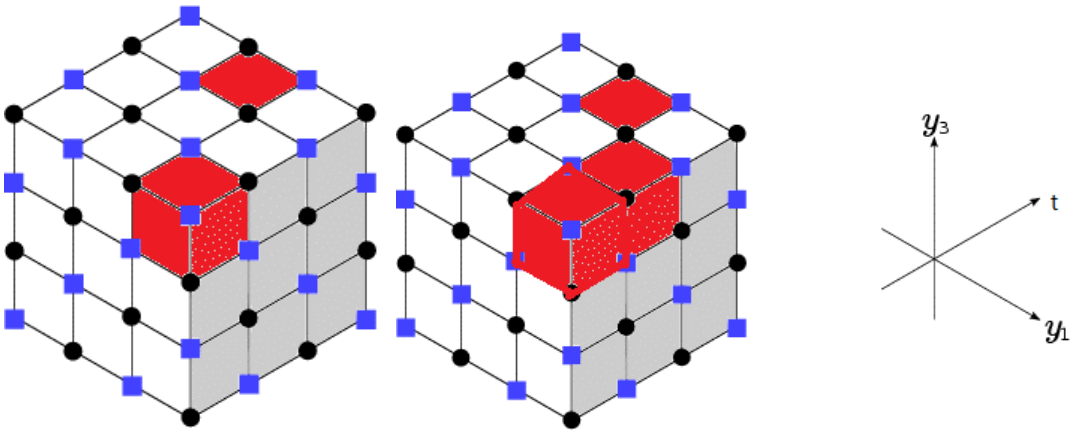


圖 2-15 時域擴展地圖示意圖 左：不合理 右：合理

其中，左圖為不合理的時域擴展地圖，其紅色點在某個時間消失在地圖上，而右側的地圖為合理的時域擴展地圖。同時，若一個點在同一個時間點上出現在兩個地方

doi:10.6342/NTU201802627

也是不合理的地圖，不過這不影響路徑規劃演算法。在時變地圖的路徑規畫搜索當中，比起單純 N+1 維地圖多了許多限制。

透過結合適應動態環境的 A*演算法與擴展卡爾曼濾波器，我們可以處理動態環境避障與路徑規劃問題。

doi:10.6342/NTU201802627

第3章 硬體架構與系統整合

在實驗部分，無人載具的架構大致可分為計算控制、感測器、動力系統等部分。

3.1 硬體架構

本實驗選用「利基應用科技公司」的客制化載具平台。載具前面兩輪為驅動輪，後兩輪為從動輪，載具的姿態改變方式為驅動輪以左右輪之速度差。載具上的硬體設備包括 ARMINNO 主控板、無線通訊模組 Zigbee、紅外線感測器、超音波感測器、電子羅盤(本實驗未使用)以及馬達驅動模組，另外再整合 AXIS 的 M3006V 型網路攝影機、微軟的 Kinect 感測器與筆記型電腦即為實驗整體的硬體架構。接著將介紹各感測器之用途和作用原理。圖 3-1 為載具的硬體架構示意圖。

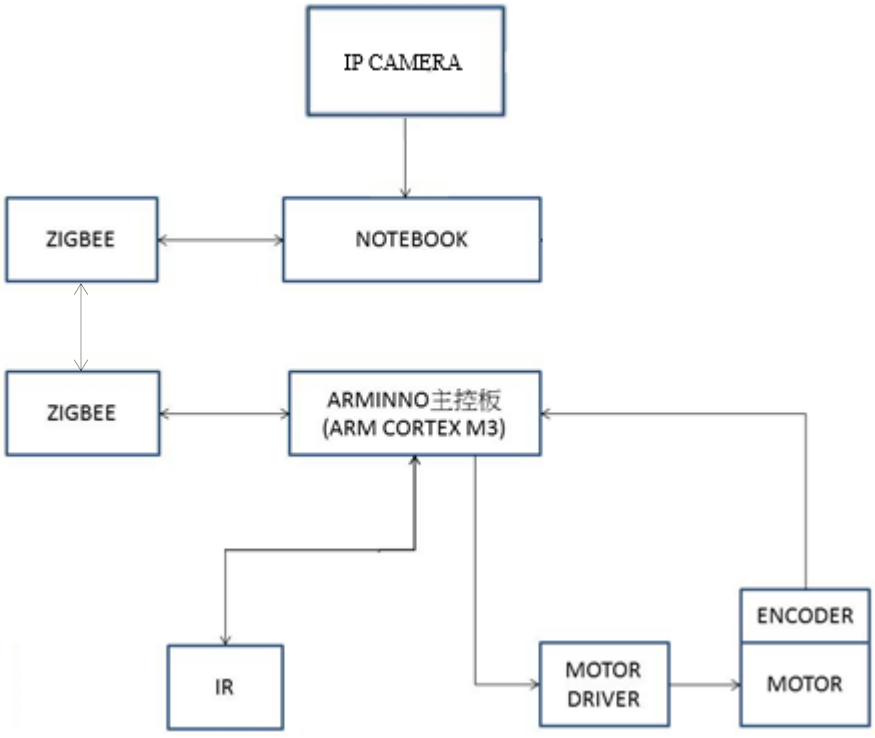


圖 3-1 載具硬體架構[11]

doi:10.6342/NTU201802627

圖 3-2 客製化載具平台[11]

3.1.1 **M3006V 型網路攝影機**

攝影機方面使用 AXIS 生產的 M3006V 型網路攝影機如圖 3-3，詳細規格如表 3-1。

圖 3-3 M3006V 網路攝影機[11]

doi:10.6342/NTU201802627

表 3-1 攝影機規格表[11]

影像感應器	逐行掃描 RGB
鏡頭	1.6 公釐
視角	134 度
燈光敏感度	0.6-200000 lux
解析度	2048x1356 到 160x90
每秒畫格	30 FPS
同步方式	Internal

3.1.2 馬達編碼器(Encoder)

左右輪馬達內安置的編碼器能使載具在運動中能夠知道自己當下的位置資訊來達成載具定位與定向。在得到左右輪前進距離後，我們以式(3.3)計算出載具的位置及姿態[45]：

$$\Delta\theta = \frac{\widehat{S_R} - \widehat{S_L}}{2b} \tag{3.1}$$

$$x_c(k+1) = x_c(k) + \left(\frac{\widehat{S_R} + \widehat{S_L}}{2}\right) \cdot \frac{2 \cdot sin\frac{\Delta\theta}{2}}{\Delta\theta} \cdot cos\left(\theta_c(k) + \frac{\Delta\theta}{2}\right) \tag{3.2}$$

$$\theta_c(k+1) = \theta_c(k) + \Delta\theta \tag{3.3}$$

其中$x_c(k)$、$y_c(k)$、$\boldsymbol{\theta_c(k)}$為載具當下的位置和姿態，$\widehat{S_R}$為右輪前進距離，$\widehat{S_L}$為左輪前進距離，$\Delta\theta$為載具姿態角差，b 為驅動輪至載具對稱軸距離。

圖 3-4 為載具定位示意，其中，O 為瞬時曲率中心，l為曲率半徑。

doi:10.6342/NTU201802627

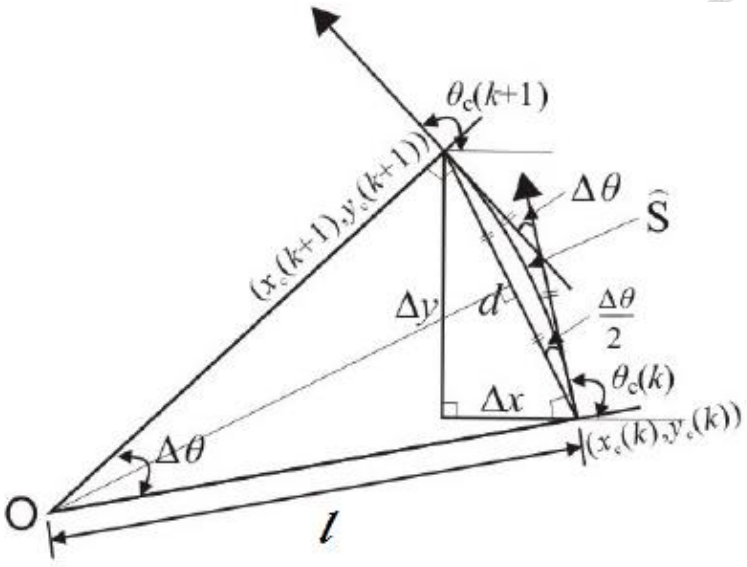


圖 3-4 載具定位示意圖[11]

3.2 載具運動方程式[11]

本實驗使用的載具為前輪驅動，兩輪方向固定，藉由調整驅動輪之轉速差改變載具的速度與姿態，後兩輪則為輔助支撐和滾動用之惰輪，如圖 3-5 所示。

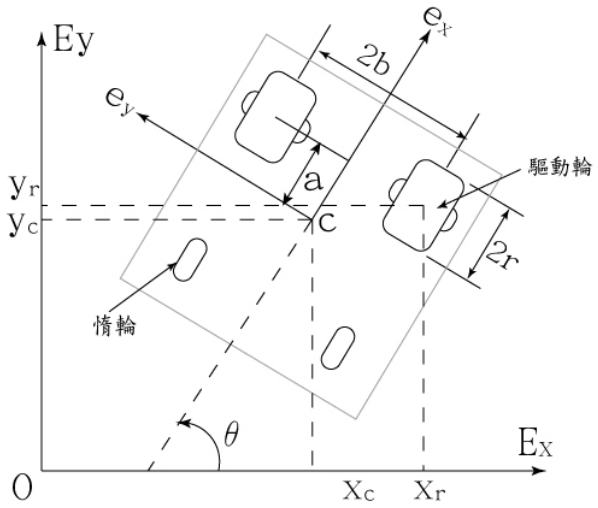


圖 3-5 載具模型示意圖[11]

其中：

doi:10.6342/NTU201802627

a :驅動輪輪軸至載具中心距離　b :驅動輪至載具對稱軸距離

r :驅動輪半徑　θ :載具相對於慣性參考座標的姿態角

$\dot{\varphi}_l$、$\dot{\varphi}_r$:分別為驅動輪左、右輪的轉速

$\mathbf{r_l}$、$\mathbf{r_c}$、$\boldsymbol{r_r}$:分別為載具左輪、質心及右輪於慣性座標的位置向量

由載具的幾何關係可得：

$$\mathrm{x(t)} \in R^n \approx x' \in R^{n+1} \tag{3.4}$$

對上式微分一次，並轉換到載具座標，可得載具速度：

$$\begin{aligned} \dot{\boldsymbol{r}}_{\boldsymbol{r}} &= \dot{\boldsymbol{r}}_{\boldsymbol{c}} + b\dot{\theta}\boldsymbol{e_x} + a\dot{\theta}\boldsymbol{e_y} \\ &= \left(\dot{x}_c cos\theta + \dot{y}_c sin\theta + b\dot{\theta}\right)\boldsymbol{e_x} + \left(\dot{y}_c cos\theta - \dot{x}_c sin\theta + a\dot{\theta}\right)\boldsymbol{e_y} \end{aligned} \tag{3.5}$$

假設輪子為純滾動而且無側滑動：

$$\dot{x}_c cos\theta + \dot{y}_c sin\theta + b\dot{\theta} = r\dot{\varphi}_r \tag{3.6}$$

$$\dot{y}_c cos\theta - \dot{x}_c sin\theta + a\dot{\theta} = 0 \tag{3.7}$$

定義載具之線速度$\boldsymbol{v}$和角速度為：

$$v = \dot{x}_c cos\theta + \dot{y}_c sin\theta \tag{3.8}$$

$$w = \dot{\theta} \tag{3.9}$$

將$\boldsymbol{v}$和$\boldsymbol{w}$帶入式(4.3)後可得右輪轉速$\dot{\varphi}_r$與載具線速度及角速度的關係：

$$r\dot{\varphi}_r = v + bw \tag{3.10}$$

依據同理也可知左輪轉速與$\boldsymbol{v}$和$\boldsymbol{w}$之關係：

$$r\dot{\varphi}_l = v - bw \tag{3.11}$$

將式(6.7)和(6.8)以矩陣表示為：

$$\begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{3.12}$$

doi:10.6342/NTU201802627

3.3 載具路徑追蹤控制[11]

3.3.1 模糊控制簡介

模糊控制是利用模糊邏輯來設計控制器的工作原理，而所謂模糊邏輯是將人類思維中難以精確定義的形容詞，如長短、大小、遠近的程度給予數學量化，讓我們更能表達想要呈現的資訊。也因此模糊邏輯被廣泛的應用在各個領域，如決策判斷、人工智慧等等。

模糊控制器的輸入和輸出是以語言化的控制規則為其主體，因此模糊控制系統首先將受控系統得到的數值進行模糊化的動作(Fuzzification)，接著根據知識和專家經驗訂定規則庫並進行模糊推論，最後再將推論出的語言項變數做解模糊化(Defuzzification)動作，得到輸出給受控系統，如圖 3-6 所示。

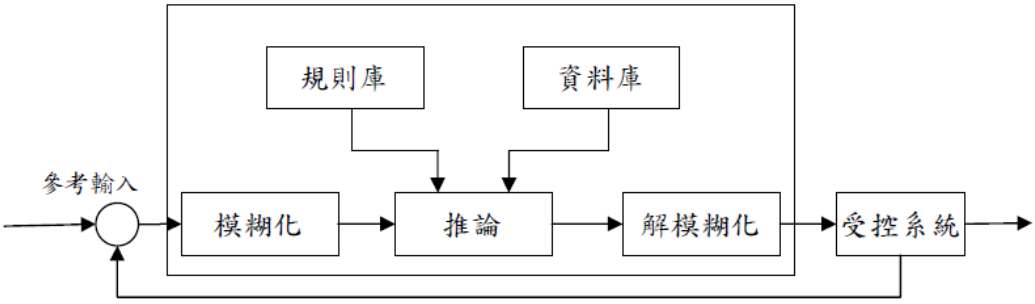


圖 3-6 模糊控制器基本架構[11]

3.3.2 路徑追蹤控制[11]

在路徑點的追蹤上，需計算出載具所需的線速度及角速度，接著由上節的載具運動方程式轉為左右輪的轉速輸出，這邊將設計三輸入兩輸出之模糊控制器。

在設計路徑追蹤的控制器時，首先要決定輸入及輸出變數。由上一節可知輸出變數為載具的線速度(v)和角速度(w)，輸入變數則參考[11]以載具與參考路徑的相對距離(d_e)、視線角(ϑ_e)、姿態角差(θ_e)為模糊控制的三個輸入變數，如圖 3-7 所示。

doi:10.6342/NTU201802627

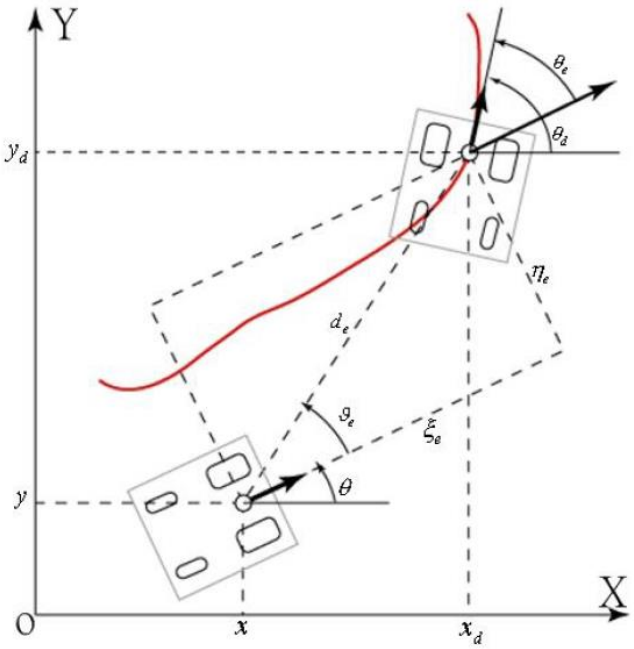


圖 3-7 參考路徑與載具姿態示意[11]

上圖中，(x, y, θ) 為載具目前的位置與姿態，(x_d, y_d, θ_d)為參考路徑點上的位置與姿態，而d_e為相對距離、ϑ_e為視線角、θ_e為姿態角差，定義如下式

$$d_e = \sqrt{{\xi_e}^2 + {\eta_e}^2} \tag{3.13}$$

$$\vartheta_e = tan^{-1}\left(\frac{\eta_e}{\xi_e}\right) \tag{3.14}$$

$$\theta_e = \theta_d - \theta \tag{3.15}$$

其中ξ_e、η_e及θ_e可由載具座標系與慣性參考座標系間的轉換求出

$$\begin{bmatrix}\xi_e \\ \eta_e \\ \theta_e\end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_d - x \\ y_\mathrm{d} - y \\ \theta_d - \theta\end{bmatrix} \tag{3.16}$$

而輸入變數及輸出變數的語言項之歸屬函數，皆選用常見並容易計算的三角型，如圖 3-8~3-12 所示。

doi:10.6342/NTU201802627

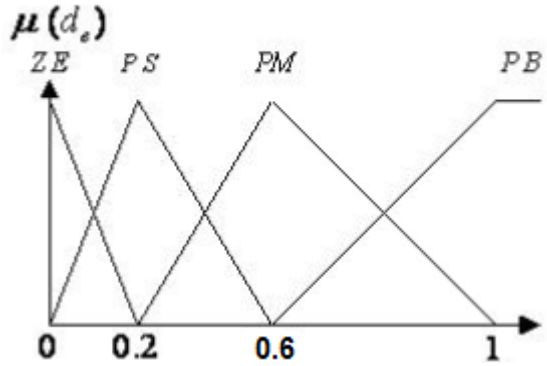


圖 3-8 輸入變數d_e之歸屬函數[11]

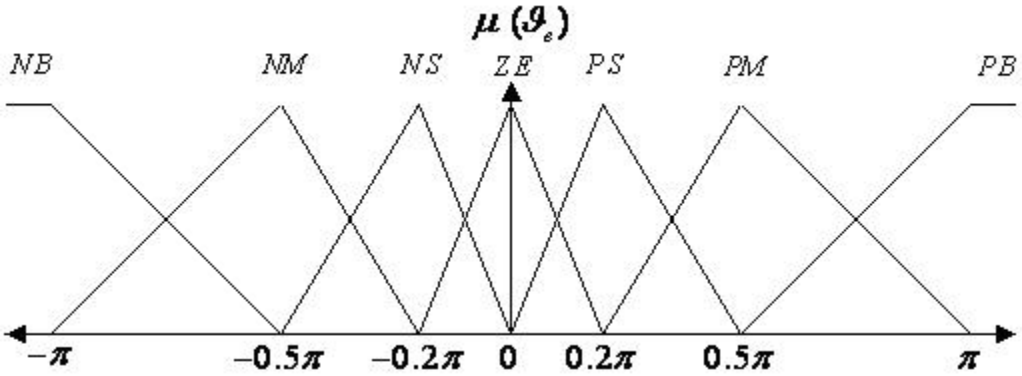


圖 3-9 輸入變數ϑ_e之歸屬函數[11]

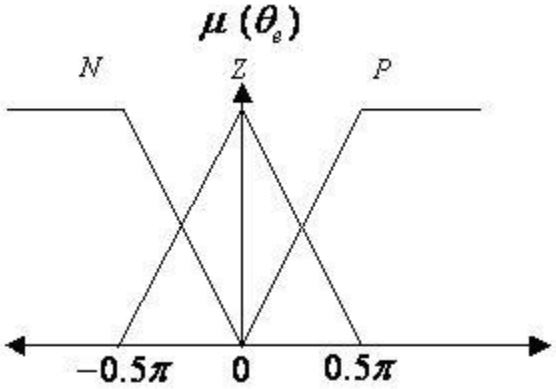


圖 3-10 輸入變數θ_e之歸屬函數[11]

doi:10.6342/NTU201802627

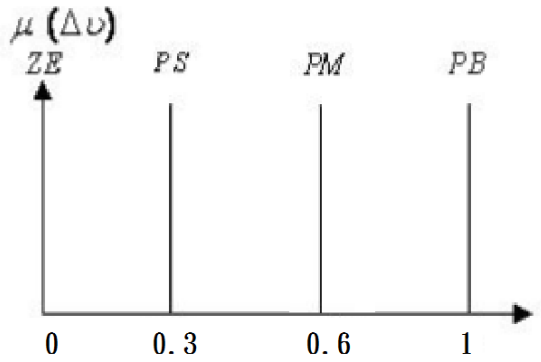


圖 3-11 輸出變數△V 之歸屬函數[11]

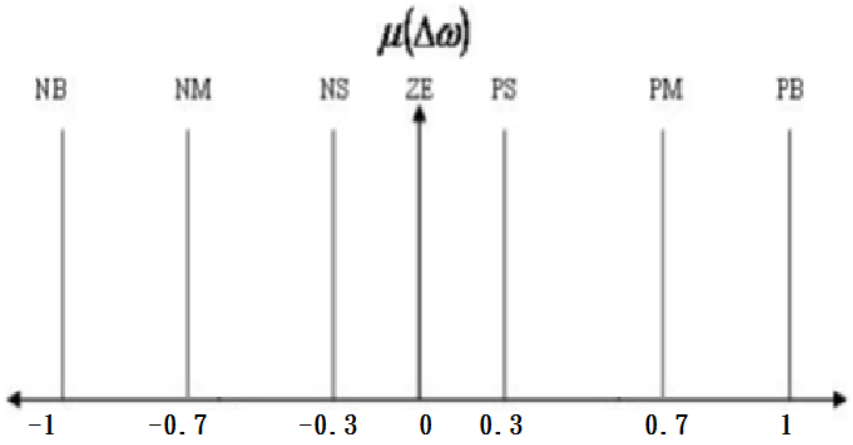


圖 3-12 輸出變數△W 之歸屬函數[11]

由於每個歸屬函數的操作區間與實際實驗時並不一定相同，因此會依實驗需求乘上正規化係數。

定義完各個輸入輸出的歸屬函數後，接著需建立模糊規則庫。原則為當相對距離和視線角過大時，代表載具遠離參考路徑，需給予較大的角速度使載具回到期望的參考路徑上；若相對距離大但視線角小，給予較大的線速度即可。並依載具之姿態角差可調整角速度的大小來達到更流暢的控制效果。

將以上的邏輯，寫成規則庫的條目如下：

$$if\ d_e = A\ , \vartheta_e = B\ , \theta_e = C\ ,\quad then\ v = D\ , w = E \tag{3.17}$$

其中，A, B, C, D, E 為模糊規則中的語言項，分別如下所示：

A = {ZE (zero), PS (positive small), PM (positive medium), PB(positive big)}

B = {PB, PM, PS, ZE, NS (negative small), NM (negative medium),NB (negative

doi:10.6342/NTU201802627

big)}

C = {P (positive) , Z (zero), N (negative)}

D = {ZE, PS, PM, PB}

E = {PB, PM, PS, ZE, NS, NM, NB}

依此定義可建立起 84 條規則庫[29]，如表 3-1 所示，再搭配模糊交集演算利用高度法解模糊化算出輸出變數。

表 3-2　路徑追蹤模糊規則庫[11]

θ_e		d_e							
P / Z / N		ZE		PS		PM		PB	
		Δv	$\Delta\omega$	Δv	$\Delta\omega$	Δv	$\Delta\omega$	Δv	$\Delta\omega$
ϑ_e	PB	ZE / ZE	PS / PS	ZE / ZE	PM / PS	ZE / ZE	PM / PM	ZE / ZE	PB / PB
	PM	ZE / ZE	ZE / ZE	PS / ZE	PS / PS	PS / ZE	PM / PS	PM / PS	PB / PM
	PS	ZE / ZE	ZE / ZE	PS / PS	PS / ZE	PM / PS	PS / ZE	PB / PM	PM / PS
	ZE	PS / PS	ZE / ZE	PS / PS	ZE / NS	PM / PM	ZE / NS	PB / PB	ZE / ZE
	NS	ZE / ZE	ZE / ZE	PS / PS	NS / NS	PM / PS	NS / NM	PB / PM	NM / NM
	NM	ZE / ZE	ZE / NS	PS / ZE	NS / NM	PS / ZE	NM / NM	PM / PS	NB / NB
	NB	ZE / ZE	PS / NS	ZE / ZE	NM / NM	ZE / ZE	NB / NB	ZE / ZE	NB / NB

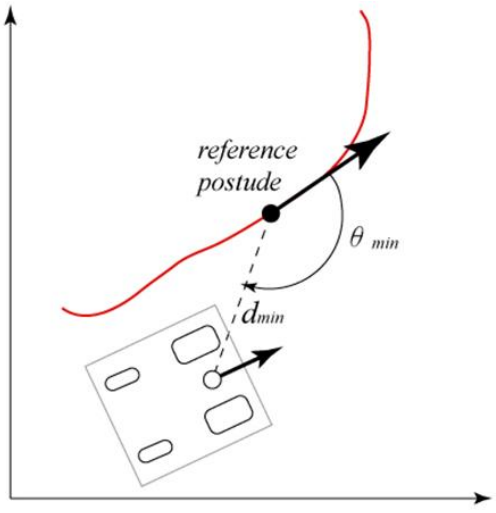


圖 3-13 最小距離閥值 d_{min} 及最小角度閥值 θ_{min}[11]

doi:10.6342/NTU201802627

第4章　模擬與實驗結果分析

4.1 模擬工具

本研究利用 c++搭配 SDL(Simple DirectMedia Layer)撰寫模擬程式。主要模擬各種線上路徑規劃，動態環境線上路徑規劃，姿態相關路徑規劃。另外，利用 python 的 pyKalman 套件撰寫利用擴展卡爾曼濾波器預測障礙物路徑的程式，然後利用 python 的 turtle Graphinc 圖形介面撰寫路徑預測模擬。最後，利用 Boost.Python 將卡爾曼濾波器加入 C++模擬當中。以下為模擬介面圖：

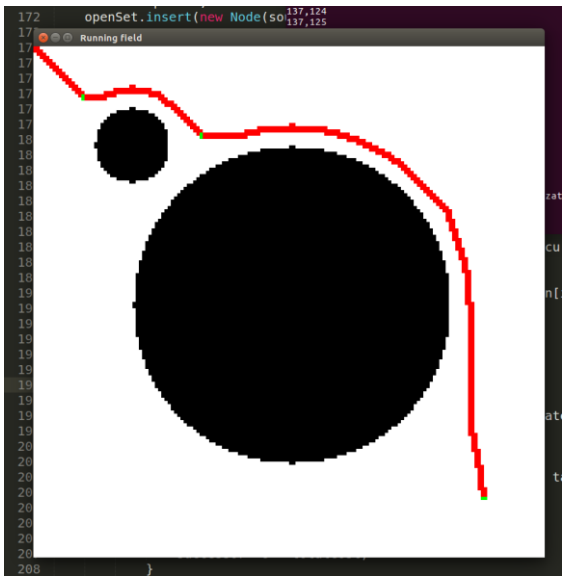


圖 4-1 線上路徑規劃模擬工具介面圖

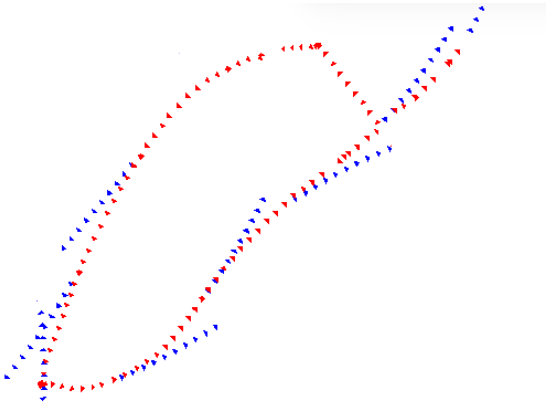


圖 4-2 路徑預測模擬介面圖

doi:10.6342/NTU201802627

4.2 模擬結果

4.2.1 線上路徑規劃模擬結果

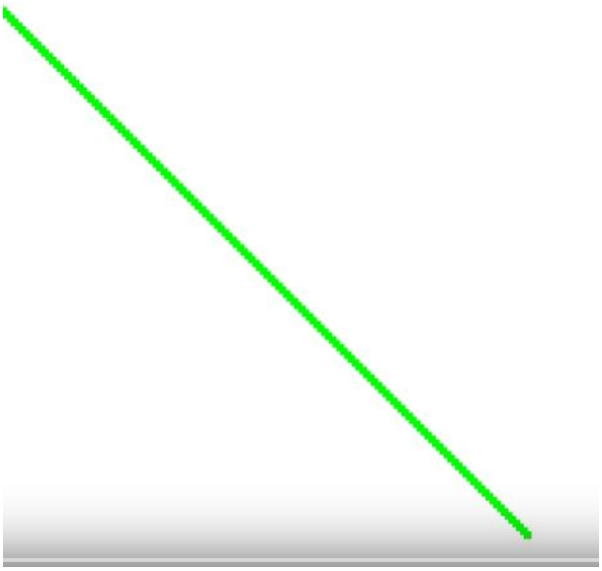


圖 4-3 線上路徑規劃模擬 1

圖四-3 表示由離線路徑規劃完成的初始路徑。由於還未有任何障礙物被觀測到，因此規劃出的路徑為一條直線，而綠色表示還未被走過的路徑。

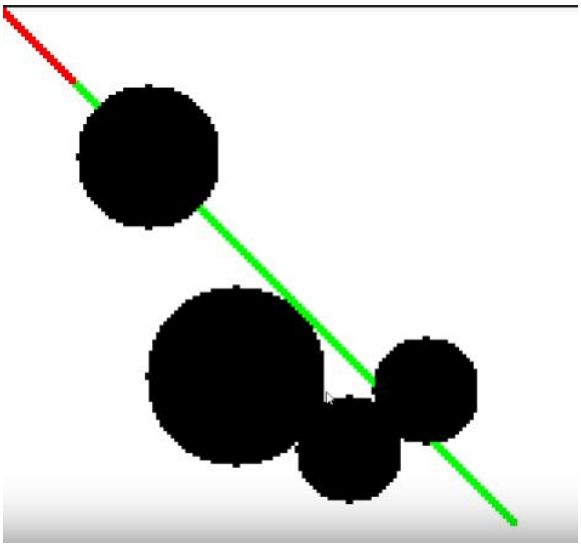


圖 4-4 線上路徑規劃模擬 2

doi:10.6342/NTU201802627

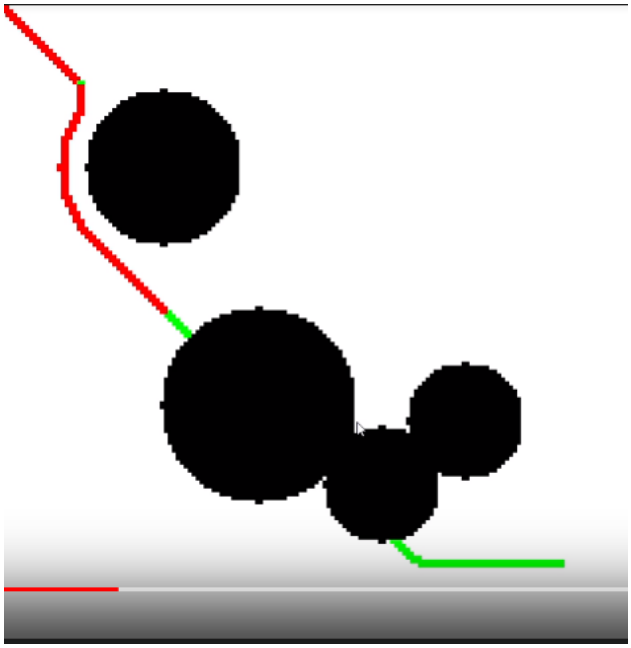
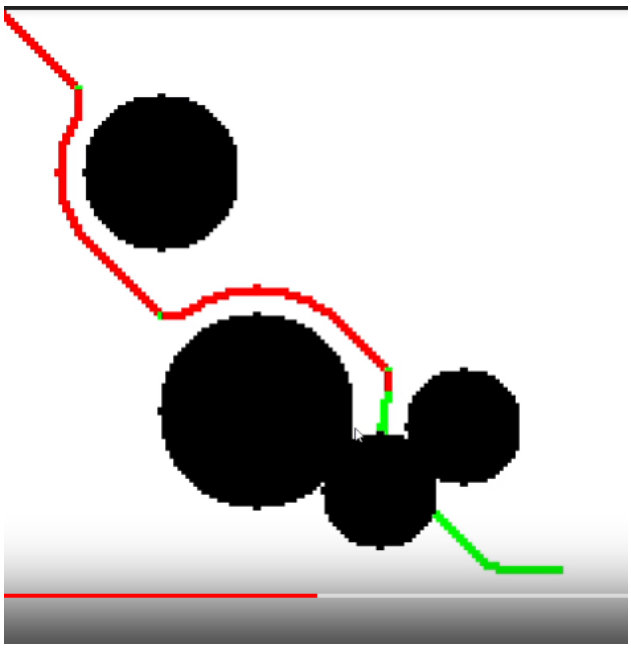


圖 4-5 線上路徑規劃模擬 3

圖四-4 和四-5 表示當載具運動到第一個障礙物的探測距離後，重新規畫路徑得到圖四-5 的新路徑。而圖四-5 中，載具運動至發現第二個障礙物後重新規畫路徑得到圖四-6 的路徑。



doi:10.6342/NTU201802627

圖 4-6 線上路徑規劃模擬 4

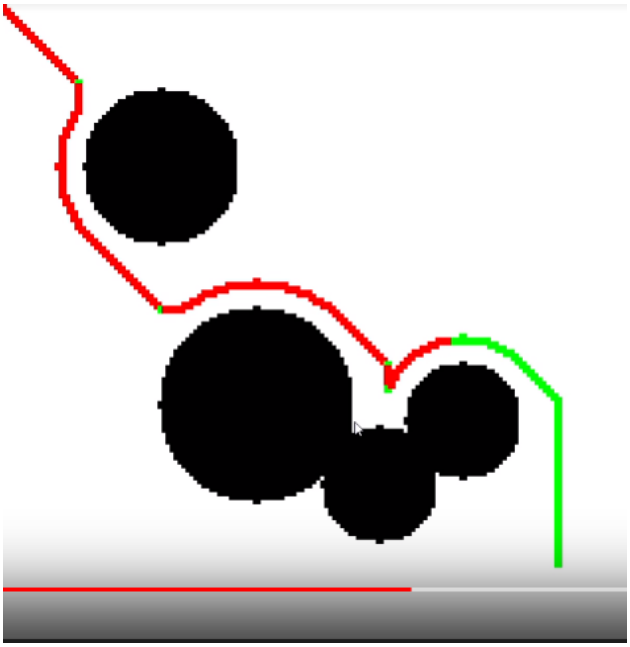


圖 4-7 線上路徑規劃模擬 5

圖 4-3 到圖 4-7 模擬了在複雜環境下的線上路徑規劃。綠色代表當前路徑，紅色代表載具已經追蹤完成的路徑，黑色代表障礙物，白色代表可以運行的區域。一開始，載具並沒有讀取到運行場域中的任何障礙物，因此由 A*演算法規劃出一條直線的路徑為抵達終點的最短路徑。然而，每當遇到新的障礙物的時候，演算法就會透過哈希函數更新地圖，並且利用適應線上路徑規劃的 A*演算法執行重新規劃的任務。此地圖大小為 1000*1000，根據 2.3.3 節所述，哈希函數如下所示：

$$\text{hashf}(x, y) = x + (L + 1) * y \tag{4.1}$$

由圖 4-6 和圖 4-7 可以知道，這個線上路徑規劃有能力面對障礙物所組成的凹洞陷阱(trap)並有能力修正路徑離開錯誤的路徑與方向。然而，實際上的情況會是一開始我們通常會得知一些地圖資訊，以下為一開始得知部分地圖資訊，也就是離線與線上路徑規劃合併的模擬。

doi:10.6342/NTU201802627

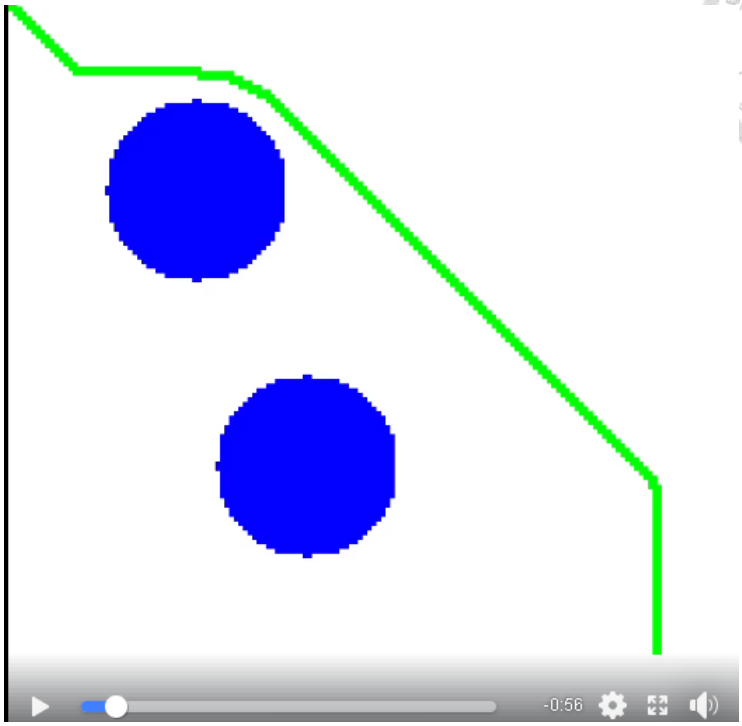
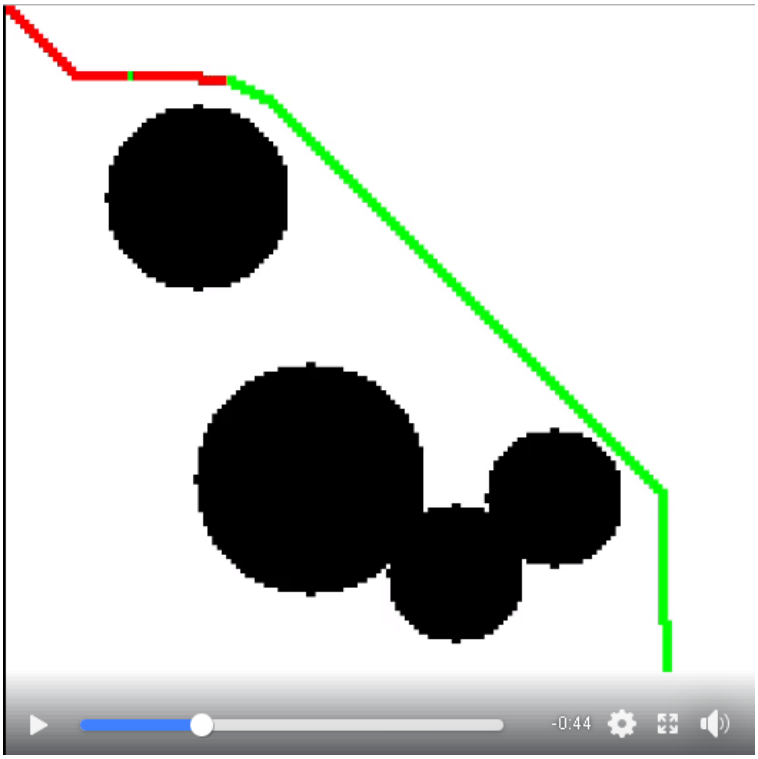


圖 4-8 合併路徑規劃模擬 1

藍色障礙物為一開始就知道的地圖資訊。而多出的黑色部分為必須進入探測範圍才可以知道的障礙物。一開始初始路徑利用 A*演算法根據已知障礙物資訊做規畫。



doi:10.6342/NTU201802627

圖 4-9 合併路徑規劃模擬 2

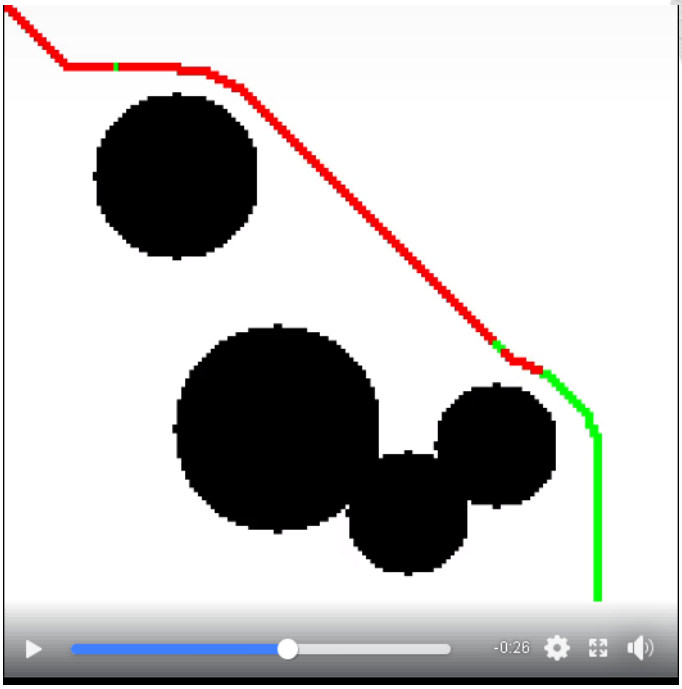


圖 4-10 合併路徑規劃模擬 3

在合併路徑規劃的模擬當中，圖 4-8 中的藍色障礙物為初始離線狀態即掌握的障礙物，而除了藍色以外的黑色障礙物為載具運行時才會發現的障礙物。由此模擬結果可知，若對地圖有初步了解，可以減少在遭遇障礙物時所需要的重新規劃。結合離線與線上路徑規畫對於部分已知地圖的規畫可以適用。

4.2.2 不同線上路徑規劃演算法運行時間比較

本論文中使用了數種方法增加傳統的 A*演算法的運行效率。第一，本論文將傳統的 A*演算法對節點資訊的傳遞與儲存做優化，並且將其透過結構上的修改使其更加適應在運行過程中的路徑規劃。本模擬將比較 A*演算法資料傳遞與儲存優化與結構優化對不同規模的問題下對時間的影響。以下為運行電腦系統的資訊表：

表 4-1 模擬環境表

	資訊
作業系統	Ubuntu Linux 16.04
CPU	intel core i7

doi:10.6342/NTU201802627

compiler	g++ std=c++11
ram	4GB

以下為比較模擬所使用的地圖。對於不同尺寸的第圖。白色部分為可運行區域，黑色部分為障礙物。

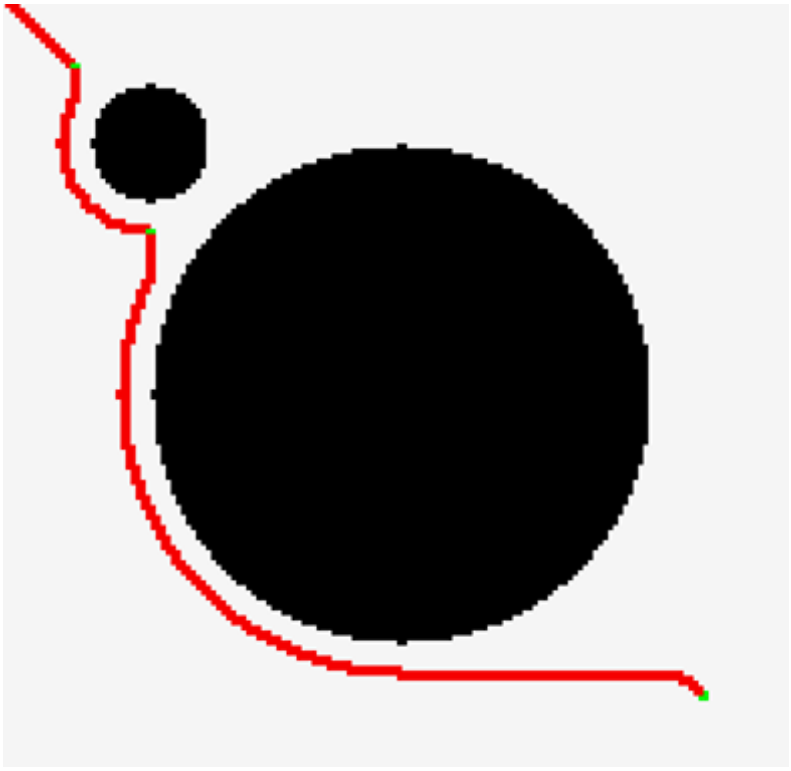


圖 4-11 時間比較模擬用地圖

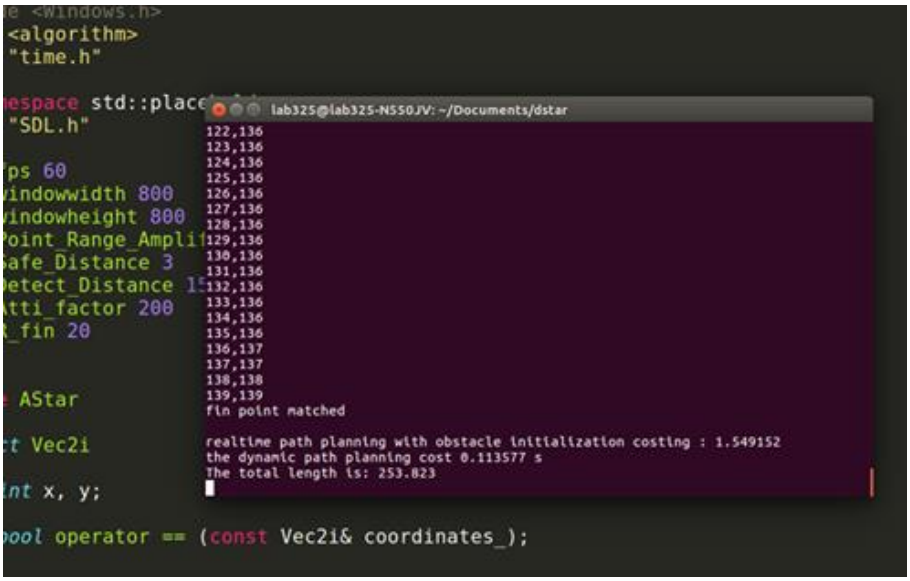


圖 4-12 模擬結果視窗示意圖

doi:10.6342/NTU201802627

以下分為三種情況進行不同規模問題(100^2 vs 1000^2)進行比較。由於 Linux 系統執行 ctime 的時候的單位為 cputime 而不是秒因此以下均以 cputime 為單位。

1. Original A*: 直接使用 A*演算法用於線上路徑規劃，如 2.3.1 節流程圖所示
2. First two modification：修改障礙物傳遞方式，傳遞障礙物的時候只傳遞殼狀障礙物，並且在 A*擴展方向增加八個方向。
3. Structure modification：增加三個 List 應對環境的改變，並且使用哈希表儲存節點資訊。

表 4-2 模擬結果表

	Original A* 100*100	Original A* 1000*1000	First two modification 100*100	First two modification 1000*1000	Structure modification(with hash) 100*100	Structure modification(with hash) 1000*1000
Online path planning with vertex information passing	10.6092 cputime(以下省略單位)	22.828337	3.340478	3.546138	0.552000	2.052744
Online path planning	10.5494 cputime	20.0621	3.30136	3.409456	0.113997	0.46853
vertex information passing time	0.06	2.866	0.04	0.136682	0.438	1.684214

由模擬結果可以知道，改變障礙物儲存技術可以大幅減少節點資訊傳遞存取所需要的時間。在 100*100 和 1000*1000 規模的問題下皆有顯著的改變。分別減少 33.3% 和 96.23%。透過觀察 original A* 和 First two modification A*可以發現一個特別的現象，若傳遞障礙物資訊使用殼狀的話，小地圖和大地圖障礙物傳遞時間差別不大。相對地，原先的 A*則在面對大地圖的線上路徑規劃時花了許多時間在傳遞資訊上。hash_table 也有相同的效果。由觀察 original A*和 structure modified A*可以發現不管在 100*100 和 1000*1000 規模的問題，線上路徑規劃的時間皆有明顯的改善，

doi:10.6342/NTU201802627

而離線路徑規劃的時間改善並不明顯。由此可知，改良式 A*演算法比起傳統 A*演算法更適應於部分已知地圖的線上路徑規畫。

4.2.3 動態環境路徑規劃模擬

線上路徑規劃演算法在面對動態環境，也就是隨時間變化的環境的時候，它必須將原先的線上路徑規劃演算法增加一個維度，並且將圖由原先的無向圖換為有向圖，將臨點分割成前點 和後點，然後引入 EKF(extended kalman filter)根據觀察而來的障礙物運動軌跡進行預測，再提供給線上路徑規劃演算法進行時間相關的路徑規劃。以下分別為運動軌跡預測模擬和動態環境路徑規劃模擬。

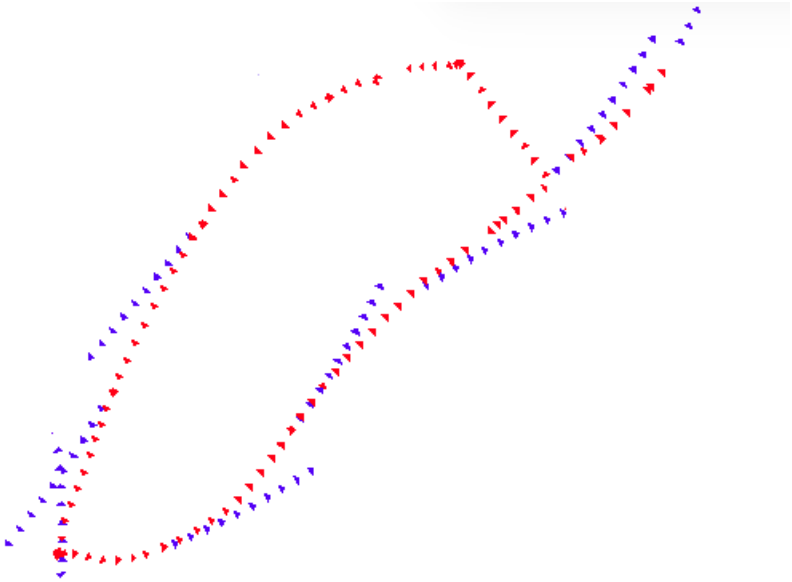


圖 4-13 路徑預測模擬結果

圖中紅色的點為實際障礙物運行路徑，而藍色部分為預測路徑，若預測路徑為同步則只顯示紅色部分。由圖可以知到，當預測離實際時間越遠的點則會得到越不準的結果，然而這些誤差可以透過增加避障時的安全距離解決。以下為透過 EKF 結合時間相關路徑規劃演算法所模擬出來的結果。

doi:10.6342/NTU201802627

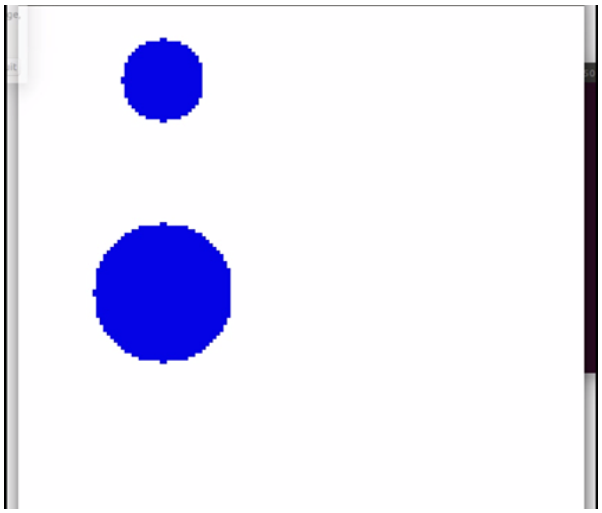


圖 4-14 動態環境路徑結果 1

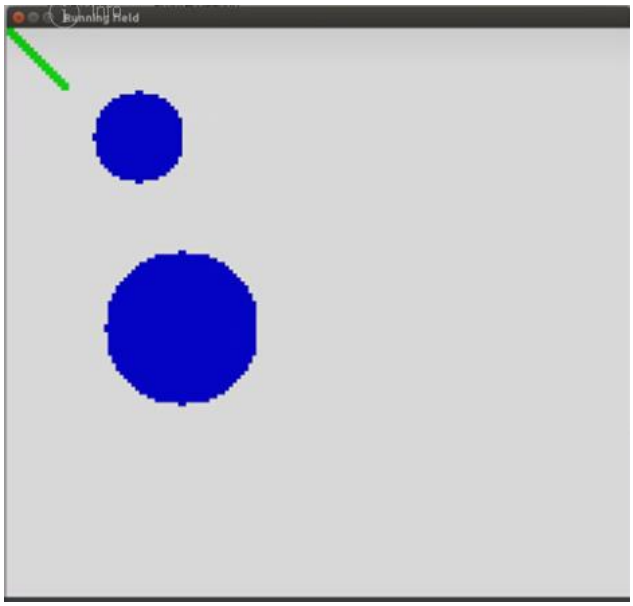


圖 4-15 動態環境路徑結果 2

doi:10.6342/NTU201802627

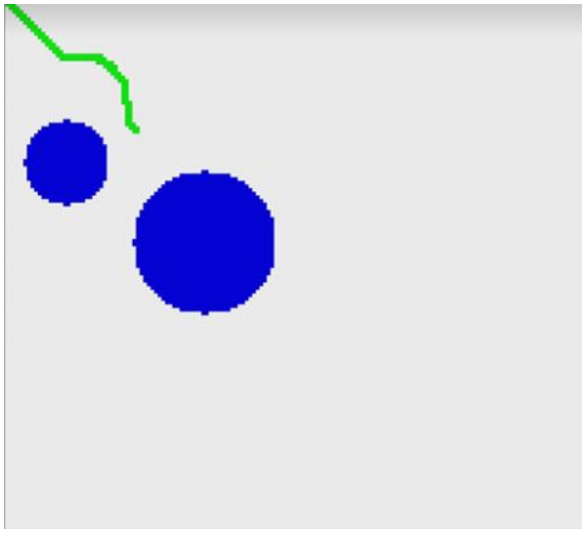


圖 4-16 動態環境路徑結果 3

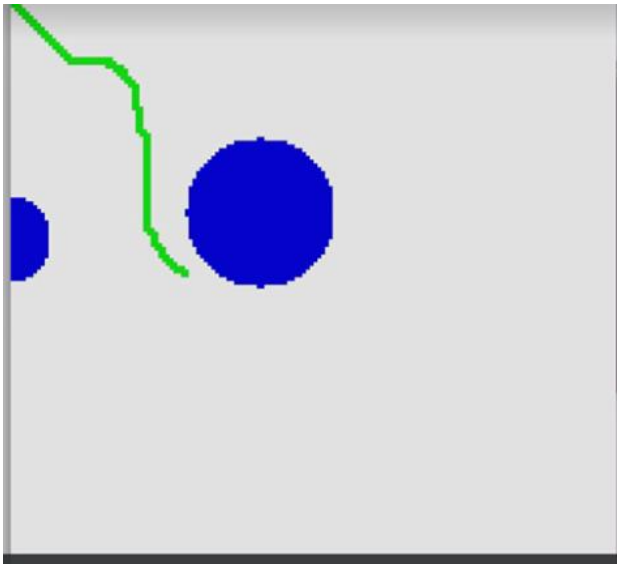


圖 4-17 動態環境路徑結果 4

其中，綠色為路徑軌跡，藍色為障礙物區域(隨時間移動)。由圖可以知道，綠色路徑在得知障礙物的運動歷史之後預測出期未來位置並提供給演算法進行路徑規劃。圖 4-15 為尚未遭遇第一個障礙物前，圖 4-16 為遭遇第一個障礙物後，圖 4-17 為遭遇第二個障礙物後。然而，第一個障礙物的運動速度為(-0.5pixel/time,0.5pixel/time)，第二個障礙物的運動速度為(-0.3pixel/time, -0.3pixel/time)，而載具每隔一個 time 前進一個 pixel。

doi:10.6342/NTU201802627

4.2.4 姿態相關路徑規劃演算法模擬

本論文中透過兩個方法解決三個路徑規劃的姿態問題，分別為：

1. 根據載具的初始姿態規劃出適應載具初始狀態的路徑
2. 在重新規畫路徑的時候納入載具的當前姿態，避免大角度旋轉
3. 要求路徑到達目標地時的末端姿態為特定姿態。

以下為模擬結果：

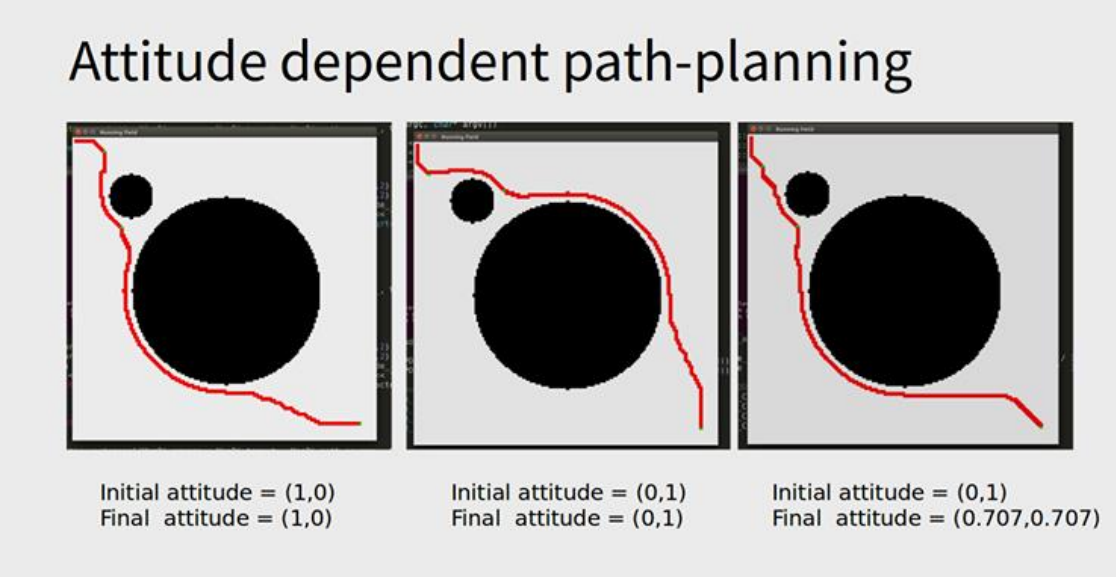


圖 4-18 姿態相關路徑規劃模擬結果

由圖 4-18 可以看出，使用姿態相關線上路徑規劃的時候，當我們要求路徑規劃演算法在到達目的時表現特定姿態時，在路徑規劃的末端就會呈現特定姿態。圖 4-18 中的左圖即要求路徑規劃演算法規劃出一條末姿態為(1,0)的路徑。中圖則要求路徑規劃演算法規劃出一條末端姿態為(0,1)的路徑。右圖則要求路徑規劃演算法規劃出一條末端姿態為(1,1)的路徑。而此演算法亦可根據載具的初始姿態規劃出不同的路徑。左圖的初始姿態為(1,0)，中圖為(0,1)。觀察一開始的路徑可以發現，初始姿態對於一開始的路徑有很大的影響力，但是一旦遠離初始點，初始姿態對於路徑的影響即顯著下降。

然而，姿態相關路徑規劃演算法也可以避免大角度旋轉。圖 4-19 為圖 4-11 雙轉彎處的放大圖，圖 4-20 為圖 4-18 右圖雙轉彎處的放大圖。前者代表原先的線上路徑規劃，後者代表使用初始姿態相關的啟發式函數的線上路徑規劃。

doi:10.6342/NTU201802627

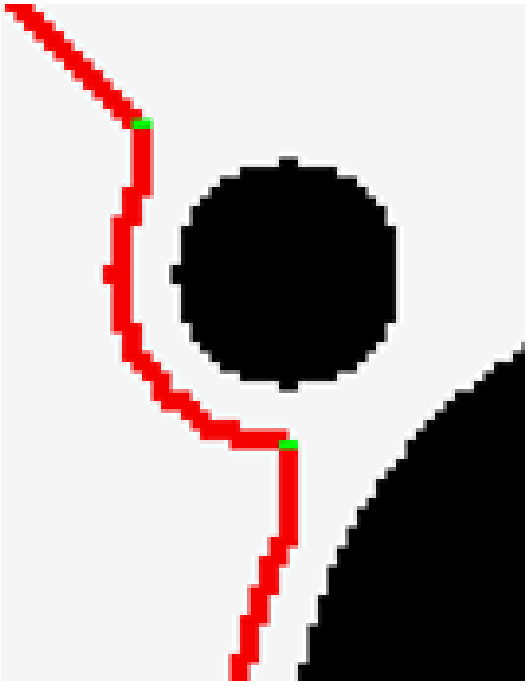


圖 4-19 雙轉彎處的放大圖

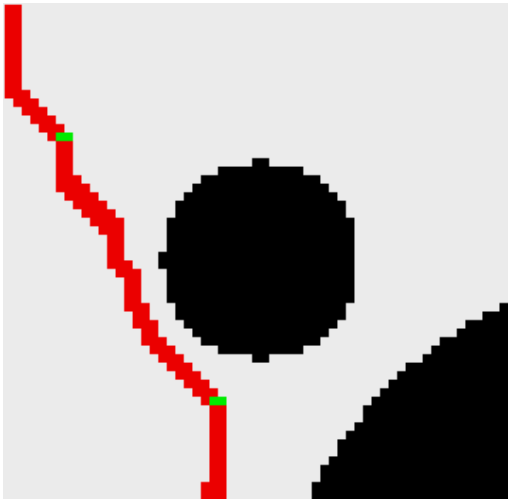


圖 4-20：右圖雙轉彎處的放大圖

比較圖 19 和圖 20 可以發現在使用路徑相關啟發式函數之後路徑會變得更加平緩，而且在需要轉彎的地方會傾向多次偏向的方式緩慢轉彎，而不是像圖 19 一樣一次大幅度旋轉，造成控制器在追蹤路徑的時候的困擾。

doi:10.6342/NTU201802627

4.3 實驗架構

本實驗透過結合模糊控制，影像辨識系統(cascade filter)[17]，路徑規劃演算法而成。其中。相機的運作下得知障礙物與車輛的位置並傳遞給路徑規劃演算法進行路徑規劃。

4.4 線上路徑規劃演算法實驗

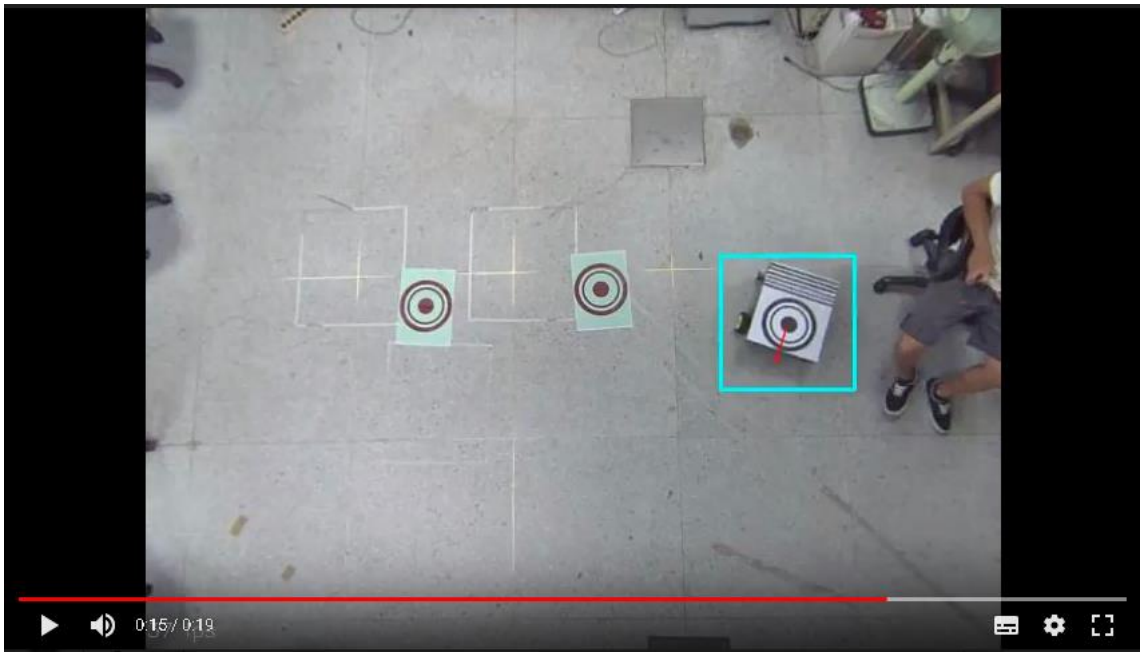


圖 4-21：實驗空照圖

圖四-21 為線上路徑規劃演算法實驗的空照圖，淺藍色框線內為載具的範圍，而紅色箭頭為姿態，而在地板上的同心圓圖形為障礙物的位置。此為載具運行到路徑末端時所拍攝的。以下為路徑追蹤圖，將充分呈現路徑規劃演算法所規劃出的路徑變化史。

doi:10.6342/NTU201802627

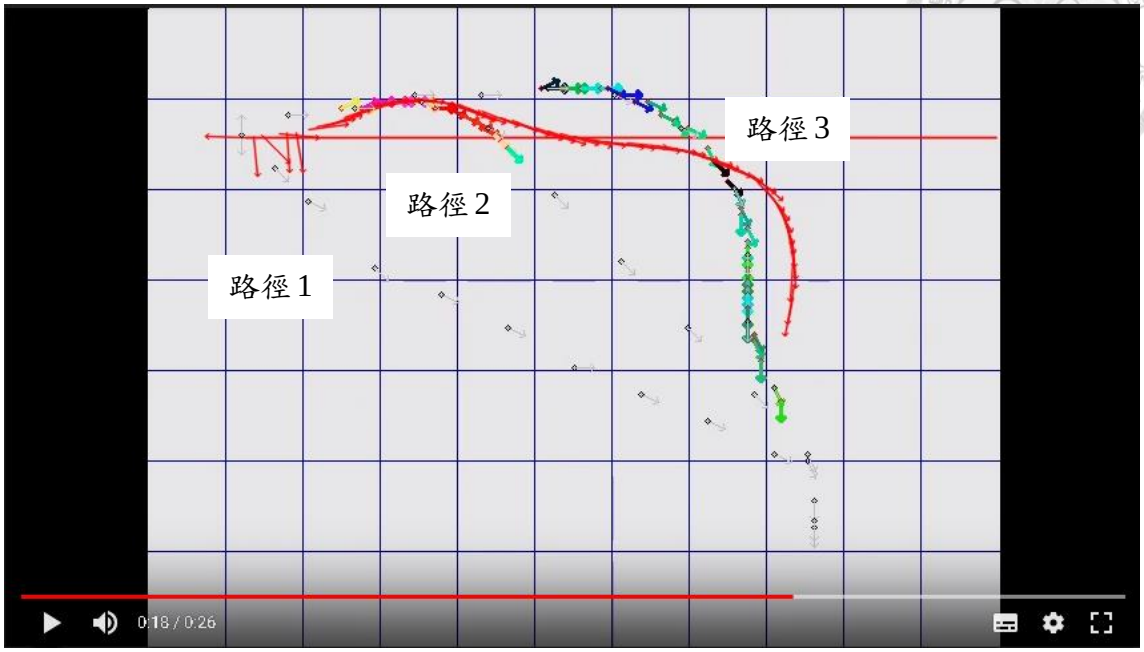


圖 4-22：路徑追蹤圖

由路徑追蹤圖可以發現一開始載具追蹤路徑一時很快地發現第一個障礙物將會導致路徑一失效，因此更新地圖後重新規劃出路徑二，接著運行到一半時發現第二個障礙物會造成路徑失效，因此重新規劃出第三條路徑。而紅色路徑為載具實際運行的位置。由圖 21 可以發現其初始姿態為(1,0)演算法會透過相機傳回來的初始姿態進行路徑規劃設計，然而本次實驗的末姿態規定為(0,1)。以下將比較不同初始姿態與末姿態的實驗圖。

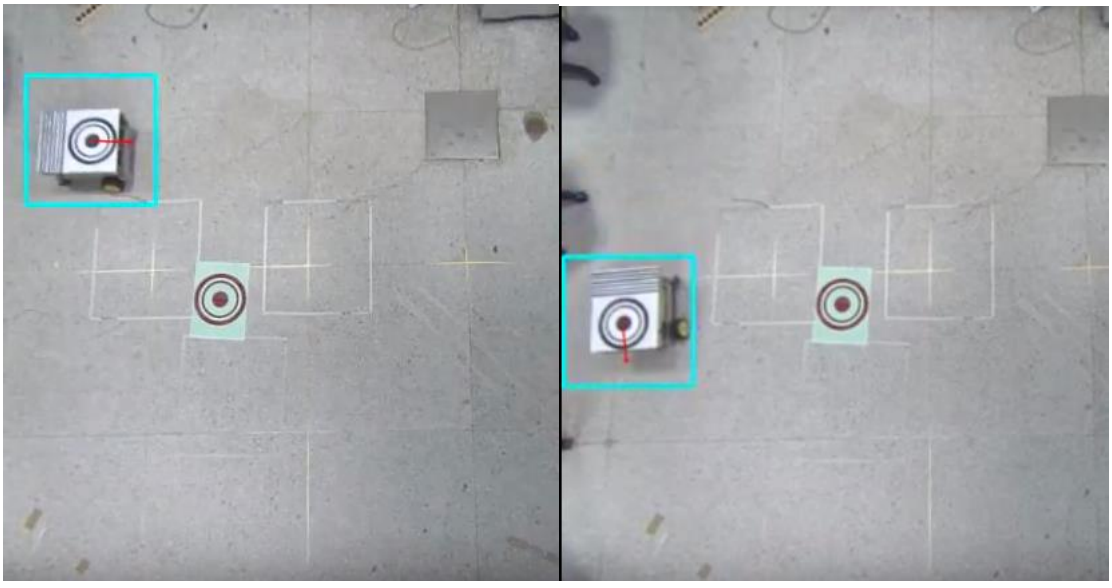


圖 4-23：*左*:初始姿態(1,0), *右*:初始姿態(0,1)實驗圖

doi:10.6342/NTU201802627

由圖 4-22 可以知道，當我們將車子擺放不同姿態的時候，演算法可以透過讀取的初始姿態替車子規劃出適應初始姿態的路徑，這兩個情況其初始點和目標點的位置均相同，只是擺放的姿態不同即有相應不同的路徑。以下為這兩種狀況的路徑追蹤圖

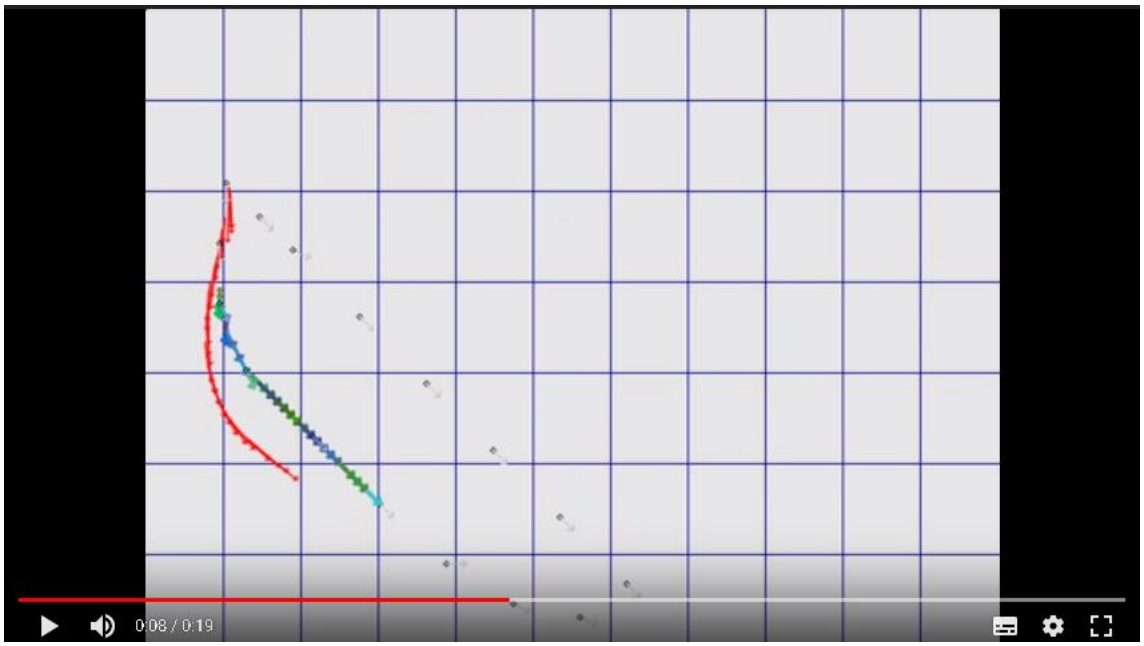


圖 4-24：初始姿態(0,1)實驗路徑追蹤圖

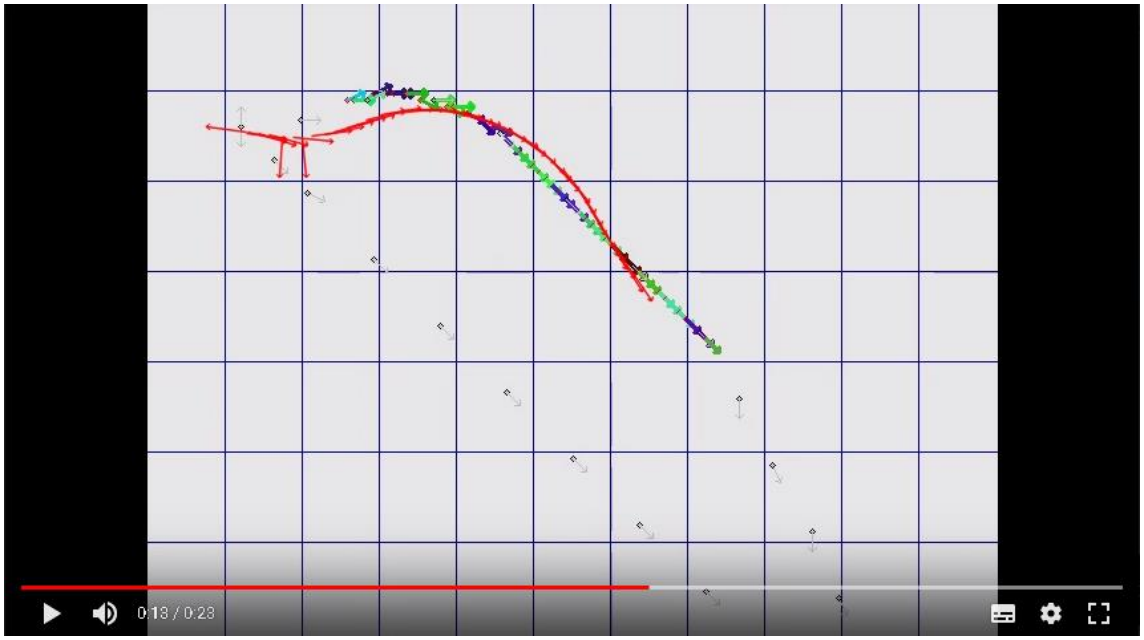


圖 4-25：初始姿態(1,0)實驗路徑追蹤圖

doi:10.6342/NTU201802627

除了初始姿態，能夠決定末姿態的路徑規劃演算法也有其實用性，以下為決定不同末端姿態的實驗比較。本論文總共處理三種不同末端姿態，分別為(0,1), (1,0), (1,1)。以下為其實驗圖：

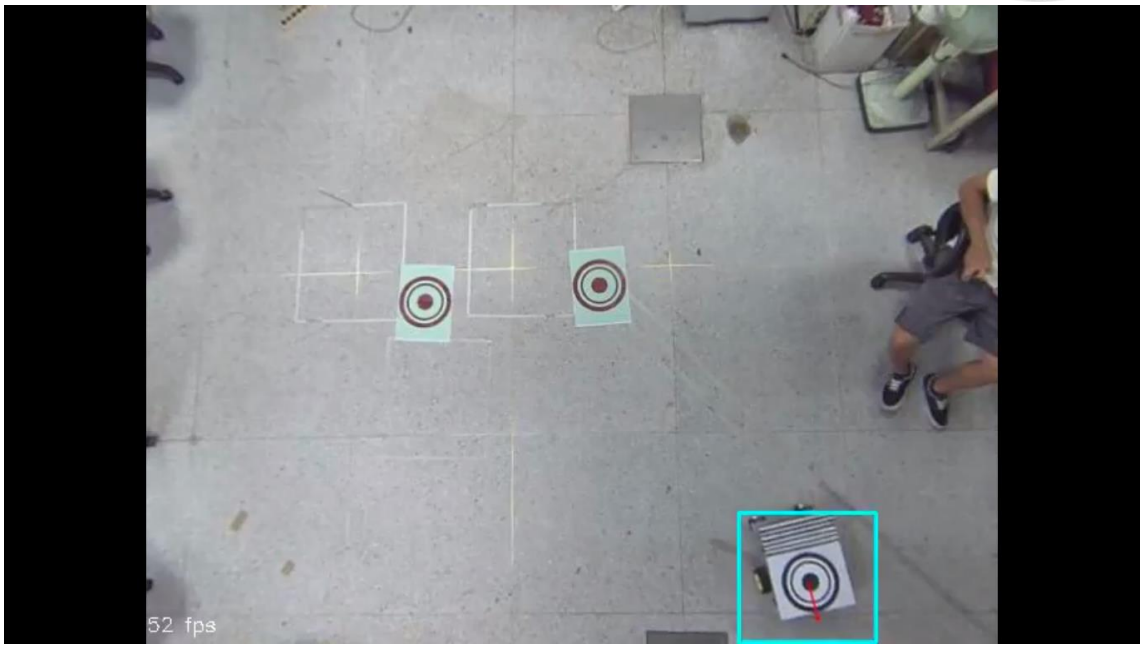


圖 4-26：決定末姿態(0,1)實驗圖

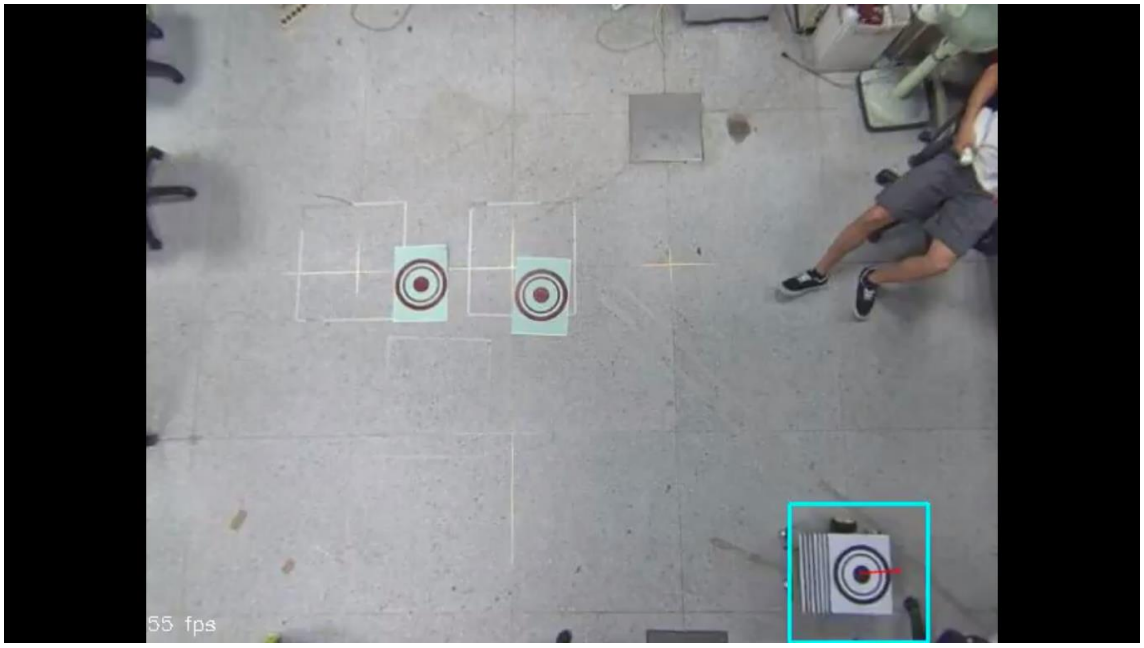


圖 4-27：決定末姿態(1,0)實驗圖

doi:10.6342/NTU201802627

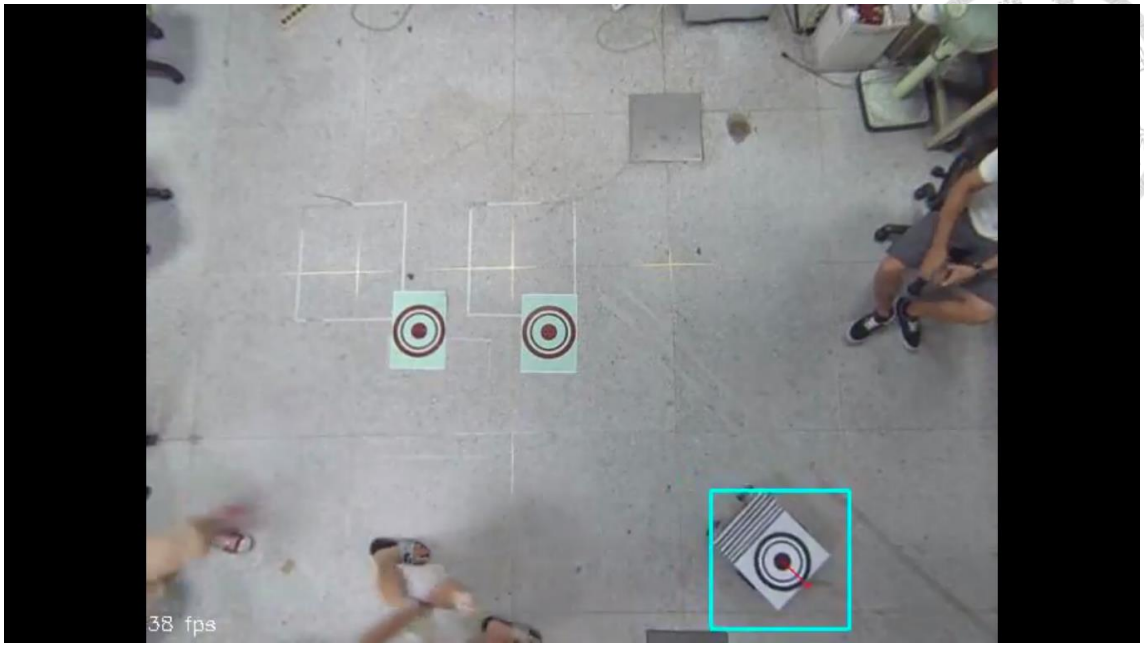


圖 4-28：決定末姿態(1,1)實驗圖

由實驗圖可以知道，這幾次實驗均按照所決定的末姿態進行路徑規劃，以下為這三次實驗的路徑追蹤圖。

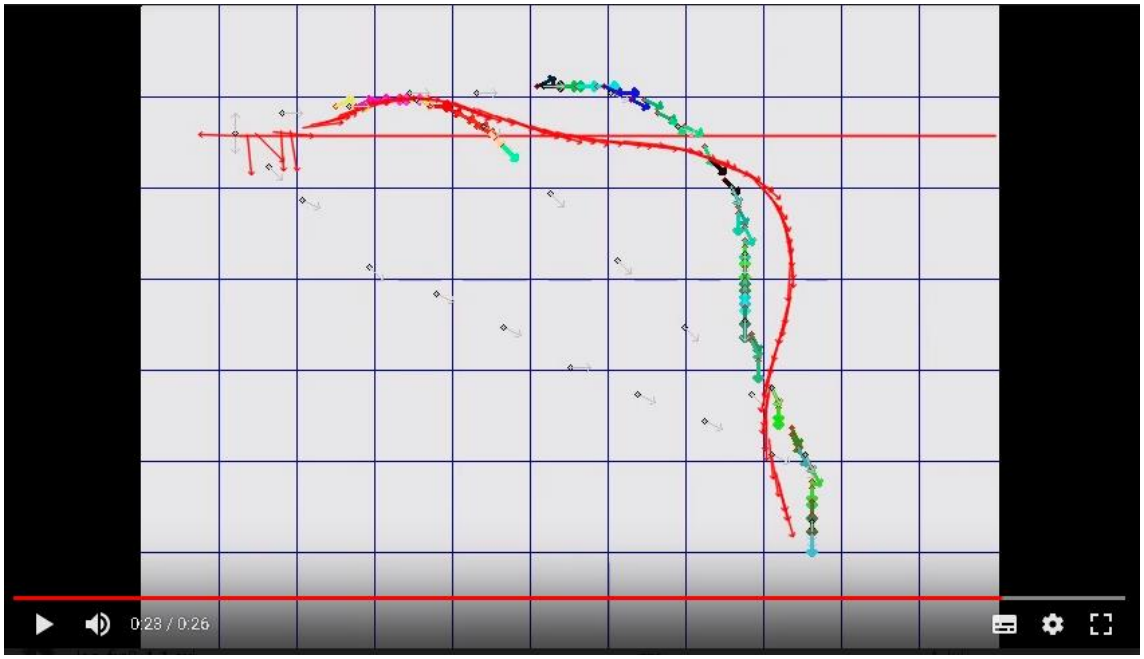


圖 4-29：決定末姿態(0,1)實驗路徑追蹤圖

doi:10.6342/NTU201802627

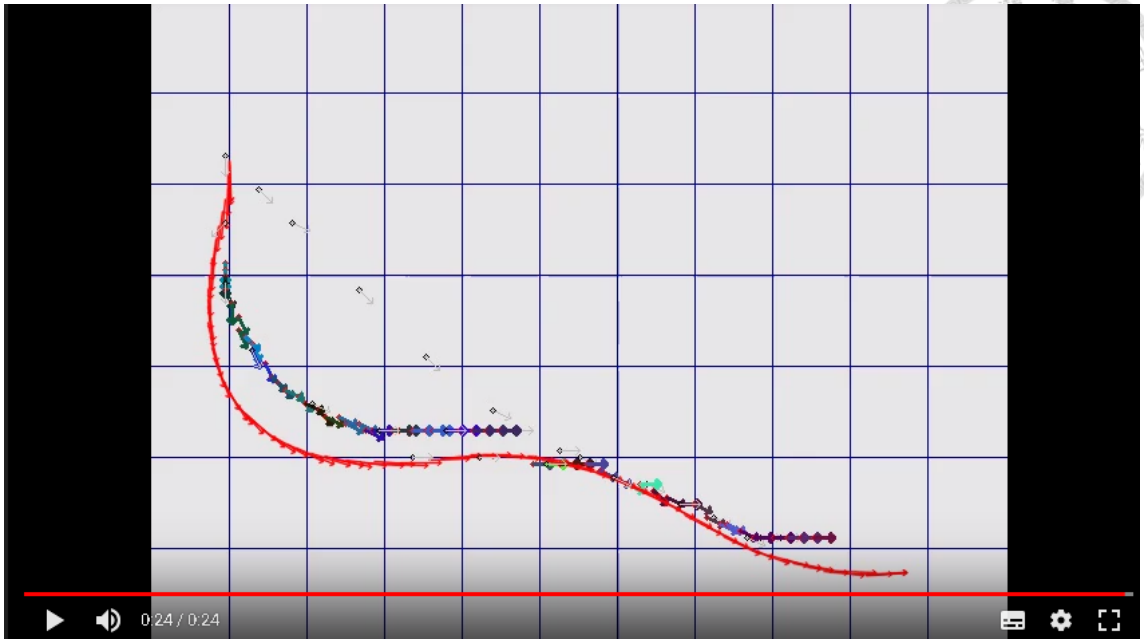


圖 4-30：決定末姿態(1,0)實驗路徑追蹤圖

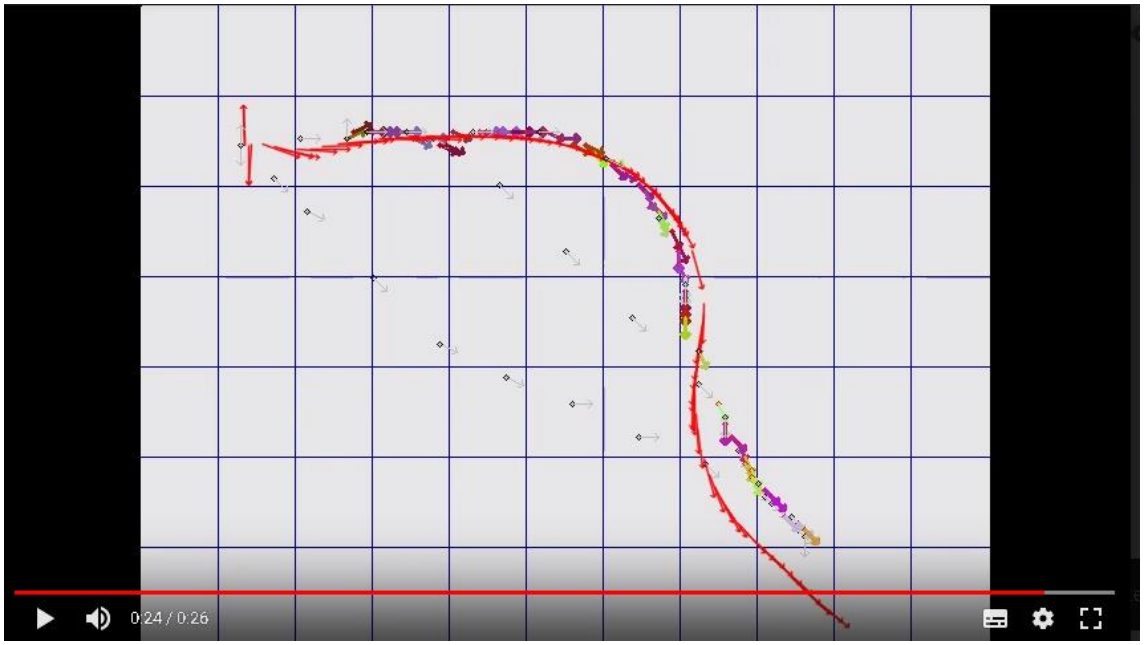


圖 4-31：決定末姿態(1,1)實驗路徑追蹤圖

其中，灰色線條為規劃出來後遇到障礙物被捨棄的路徑，彩色部分為現行路徑而紅色線條為載具的真實位置。比較圖 4-29 和圖 4-31 可以發現，在相同起始點相同終點，且初始姿態相同為(1,0)的情況下，指定末端姿態不同導致演算法對路徑做相應的修正以符合末端姿態的要求。

doi:10.6342/NTU201802627

第5章　結論與未來方向

本論文提出傳統動態規劃衍伸而來的 A*路徑規劃演算法之修正，並且使其適應現行自駕車對路徑規劃的一些要求。其中包含結合離線路徑規劃和線上路徑規劃以因應部分探索地圖問題和地圖資訊改變問題。此外，透過修正 A*演算法的障礙物傳遞方法、結構與節點資訊傳遞方法，可提升 A*演算法的效率與面對地圖改變時的應變能力。接著，透過重新設計啟發式函數與調整動態終點的方法使路徑規劃演算法得以規劃出符合初始狀態、符合末端姿態要求、且避免大幅度轉向的路徑。最後，透過增加時間維度與將臨點分割成前點 與後點的概念，處理時間不能回頭的問題，並且以此將三維路徑規劃等效成隨時間改變環境的二維路徑規劃，再利用 EKF 預測觀察的障礙物位置，並且將這兩者結合成動態環境路徑規劃演算法。在實驗部分，本研究利用結合影像辨識系統與控制器的自走車系統，進行線上路徑規劃演算法、姿態相關路徑規劃演算法等實驗。經實驗結果驗證所提出的演算法順利達成任務。本研究利用模擬檢驗 EKF 輔助之動態障礙物路徑規劃演算法。

由於三維路徑規劃演算法相當費時，難以處理大規模(如 1000*1000 地圖)的動態路徑規劃問題。未來也許可以結合分區路徑規劃演算法，在面對移動的障礙物時，可將路徑規劃到現存路徑的最接近點，這樣在面對大規模問題的時候可以減輕計算量，並且使之更有效率。

doi:10.6342/NTU201802627

參考文獻

[1] Tuncer, Adem, and Mehmet Yildirim. "Dynamic path planning of mobile robots with improved genetic algorithm." Computers & Electrical Engineering 38.6 (2012): 1564-1572.

[2] Song, K. T., & Sheen, L. H. (2000). Heuristic fuzzy-neuro network and its application to reactive navigation of a mobile robot. Fuzzy Sets and systems, 110(3), 331-340.

[3] 王冠尹,“適應性與啟發式 RRT 演算法在無人載具導控之應用”台灣大學應用力學研究所碩士論文，中華民國一百零六年七月。

[4] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[5] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." IEEE transactions on Systems Science and Cybernetics 4.2 (1968): 100-107.

[6] Koenig, Sven, and Maxim Likhachev. "Fast replanning for navigation in unknown terrain." IEEE Transactions on Robotics 21.3 (2005): 354-363.J. Buhl, D. J. T. Sumpter, I. D. Couzin, J. J. Hale, E. Despland, E. R. Millter & S. J. Simpson,“From Disorder to Order in Marching Locusts,” Science, Vol. 312, pp. 1401-1406, 2006.

[7] Goldenstein, Siome Klein. "A gentle introduction to predictive filters." Revista de Informatica Te?rica e Aplicada 11.1 (2004): 63-92..

[8] Kalman, Rudolph E., and Richard S. Bucy. "New results in linear filtering and prediction theory." Journal of basic engineering 83.1 (1961): 95-108.P. Misra & P. Enge, Global Positioning System, Ganga-Jamuna, Lincoln, MA, 2006

[9] G.L. Smith; S.F. Schmidt and L.A. McGee (1962). "Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle". National Aeronautics and Space Administration..

[10] Stordal, A. S. (2008). Sequential Monte Carlo Methods for Bayesian Filtering (Master's thesis, The University of Bergen).

[11] Jer-Wen Huang, “Development of Global Vision Positioning System for Pseudo-Rigid Formation Control,” Graduate Institute of Applied Mechanics, National Taiwan University Master Thesis, 2016.

[12] https://en.wikipedia.org/wiki/A*_search_algorithm#cite_note-nilsson-3

[13] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

[14] https://en.wikipedia.org/wiki/Hash_table

[15] Wu, Zhenyu, and Lin Feng. "Obstacle prediction-based dynamic path planning for a mobile robot." International Journal of Advancements in Computing Technology 4.3 (2012): 118-124.

[16] Goldenstein, Siome Klein. "A gentle introduction to predictive filters." Revista de Informatica Teórica e Aplicada 11.1 (2004): 63-92.

[17] 劉禮榮,”即時全域視覺定位系統於載具控制之應用”台灣大學應用力學研究

doi:10.6342/NTU201802627

所碩士論文，中華民國一百零七年七月。

doi:10.6342/NTU201802627