Appear in *Proceedings of the Sixth Conference on Artificial Intelligence and Applications,* Nov. 2001.

以 RRTs 建立漸進學習之運動計畫器

謝揚權　　李蔡彥
國立政治大學 資訊科學系
台北市文山區指南路二段 64 號
TEL: (02)29393091 ext 62266
E-mail:{g8909,li}@cs.nccu.edu.tw

摘要

運動計畫的技術已廣泛地被應用在機器人學及電腦圖學等領域。其欲解決的基本問題是在一散佈著障礙物的場景中，給定一物體的起點與終點，再利用運動計畫的演算法來搜尋該物體由起點到終點不會碰撞到障礙物的可行路徑。這類問題的運動計畫方法可分為兩類：單一查詢及多次查詢。前者的好處是可以應用於動態的環境中，而後者的優點是可以透過對環境做事前的處理而減少搜尋所花費的時間。在本論文中，我們延展文獻中快速擴展隨機樹(RRTs)的結構，建立一種稱為 RRT-Roadmap 的資料結構，並提出新的演算法以有效解決運動計畫之問題。這個方法同時具備了單一查詢及多次查詢之方法的優點。我們已根據該資料結構及演算法實作出一個具備漸進學習之特性的路徑計畫器，分別於靜態以及動態的環境下實驗，並對實驗之結果做分析與討論。我們相信，此方法可有效提昇運動計畫器的效率及其適用的範圍。

關鍵詞：運動計畫、路徑計畫、漸進式學習、動態環境計畫、機器學習、快速擴展隨機樹（RRT），RRT-Roadmap。

1. 概論

物體運動計畫技術的發展起源於機器人學領域中為自走式機器人或機器手臂自動產生避碰的路徑。近年來，此研究所發展出的技術，也被廣泛地應用在電腦動畫、虛擬實境、電腦輔助設計、及生化製藥等領域上。其基本問題的型態是在一個場景之中，為物體計畫出一條可行的路徑，使其能夠到達所要求的地點，並且不會碰撞到場景之中的障礙物[7]。通常我們將所要計畫路徑的物體稱為 *robot*，而其所在的場景稱為工作空間(workspace)，而 *robot* 在工作空間中的確切位置則是以組態(configuration)來表示。

假設在一 2D 的工作空間裡，令 X 代表 *robot* 的組態，則其型式為 $X = (x, y, \theta)$，其中包含了 *robot* 的位置座標以及其方向角度的資訊。就此例而言，*robot* 共有三個自由度，亦即 *robot* 所有可能的組態所形成的空間為一個三度空間，我們稱之為組態空間(Configuration Space, C-space)，並記做 C；而令 C_{free} 代表 C 中不會讓 *robot* 與障礙物碰撞的所有組態所成之集合，我們稱其為自由空間(freespace)或是合法空間。

一般我們假設 *robot* 為一可自由移動的剛體，因此在 2D 的工作空間裡只有三個自由度。但是這樣的假設並非必要，亦即 *robot* 的自由度可以超過三個，並且會隨著工作空間的維度一起增加；如 3D 的工作空間裡，自由移動的物體至少會有六個自由度，而運動計畫時的組態空間也會跟著調整。

運動計畫問題的解決方法可以分為兩類：單一查詢(single-query)以及多次查詢(multiple-query)。單一查詢是指事先給定 *robot* 的起點及終點的組態，透過路徑計畫器的運算，可以求得一條可行的路徑；而每一次的查詢都屬於獨立的問題，並且不需要進行任何的事前處理程序(preprocessing)。多次查詢則是假設所要解決的問題都是在相同的環境之下，所以可以根據場景中現有的資訊，事先建立好一種稱為「街圖(roadmap)」的資料結構，以便進行任意次數的查詢，以減少每次運動計畫所需的運算時間。單一查詢的好處是不需要事前的處理程序，所以可以應用在動態的環境之下；而其缺點是每一次查詢都必須獨立的運算，需要花費較長的時間。而多次查詢的優點是可以重複利用事先建立好的街圖來提昇運動計畫的效率，但是必須耗費額外的記憶體來儲存整份街圖，並且不適用於障礙物會移動的動態環境之中。

在本論文中，我們改良快速擴展隨機樹(Rapidly-exploring Random Tree, RRT)的資料結構，建構出另一種稱為 RRT-Roadmap 的資料結構。這個資料結構結合了單一查詢和多次查詢之優點：不需要事前的處理程序、運算過之結果可重複利用、及可以支援動態的環境。利用 RRT-Roadmap，我們實作出一具備漸進學習特性的路徑計畫器，並透過實驗觀察其實際應用的效果。我們亦將分析所得的結果，以做為後續改進的參考。

在第 2 節我們會先介紹此領域的相關研究，而第 3 節則詳細說明 RRT-Roadmap 的概念以及實際建立 RRT-Roadmap 的演算法。在第 4 節裡，我們加入了即時修改 RRT-Roadmap 的方式，使其可以

應用在動態的環境之下，而第 5 節是實驗的結果與分析，最後則是本篇論文之結論。

2. 相關研究

傳統解決運動計畫問題的方式可分為三種：cell decomposition、potential field 以及 roadmap。其中 potential field 的方法適用於處理單一查詢運動計畫的問題，而其他兩者則是應用在多次查詢運動計畫之上。 cell decomposition 是將自由空間劃分為許多的小區塊(cells)，再將相鄰的區塊以 graph 的形式連結在一起，然後在其上搜尋路徑[14][2]。Potential field 的方式是在組態空間中建立人工的位能場，利用位能場中的物體會有由高位能往低位能地區移動的趨勢，從中紀錄物體流動的軌跡而得到所求的路徑[5][4]。但是 potential field 的方式常會遭遇到局部最小值(local minimum)的問題，使得物體陷落在某個區域而無法跳脫。Roadmap 的方法則是在場景中建立街圖，使得自由空間形成類似網路的形式，而每一對合法的組態間都可以找到路徑互相連結[13][3]。

近年來，使用 randomized 的運動計畫方式漸漸地受到歡迎，因為這樣的方法可以適用於高維度的空間，並且也曾被用來解決 potential field 方法中局部最小值的問題。近年來這方面研究的發展可以 RRTs 做為代表[7][6]，其為一種有效率的資料結構，可被用於解決 holonomic、nonholonomic 以及 kinodynamic 的問題[9]，而我們所提出的漸進式街圖亦是屬於 RRT-based 的方法。

運動計畫的技術也被應用於電腦圖學的領域，如虛擬實境之導覽系統中，有關導覽路線的計畫[10]，亦或是虛擬照相機的運動控制[11]。此外，在虛擬實境的場景中利用運動計畫來輔助瀏覽，也可以有效地幫助使用者操控場景。

3. 漸進式街圖之建立

RRT-Roadmap 主要是利用 RRT 這種資料結構，並配合 RRT-Connect 演算法的觀念所改良而成[6][7]。在實際的應用上，利用 RRT-Roadmap 來解決問題可歸類為單一查詢運動計畫，因為其不需要任何事前的處理程序(碰撞偵測模組除外)，但是卻可以保有多次查詢運動計畫之可學習的特性：累積先前的經驗，並逐步地擴大場景中街圖之規模。這樣的作法，一方面可改進傳統的街圖式運動計畫必須事先耗費大量的時間來建立一個完整街圖的缺點；另一方面，又可以具備單一查詢運動計畫的彈性，並保留其運動計畫的結果以便往後加以重複利用。

在本節中，首先就文獻中 RRTs 以及 RRT-Connect 的觀念做介紹，接下來再詳述如何建立 RRT-Roadmap；而有關動態環境下的因應之道，則留待下一節再討論。

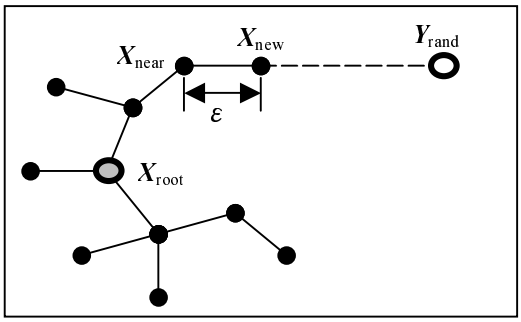

圖 1: RRT 建構之概念圖，X_{root} 為 RRT 之樹根，Y_{rand} 是 C_{free} 中隨機選取之組態，X_{near} 為 RRT 中與 Y_{rand} 最接近之節點，X_{new} 則為新加入之節點。

3.1 RRTs

Rapidly-exploring Random Tree(RRT)是一種有效率的樹狀資料結構，可以被廣泛地應用來解決許多類型的運動計畫問題，其主要的概念如圖 1 所示：T_k 代表一顆擁有 k 個節點的 RRT，且 $T_k \in C_{free}$，而 $X \in T_k$，為 T_k 之節點，其中 X_{root} 代表 T_k 的樹根。假設 Y_{rand} 為工作空間中隨機選取的一個組態，滿足 $Y_{rand} \in C_{free}$ 之條件，則我們可以根據 Y_{rand} 來找出 X_{near}，其中 $X_{near} \in T_k$ 且 X_{near} 為 T_k 的節點中與 Y_{rand} 最接近的節點；假設 s，$t \in C_{free}$，令 $Dis(s, t)$ 代表兩個組態間的幾何距離，則 X_{near} 與 Y_{rand} 的關係可以表示為：$Dis(X_{near}, Y_{rand}) \leq Dis(X, Y_{rand})$。接下來延著 X_{near} 與 Y_{rand} 的連線上試著求得 X_{new}，而 X_{new} 必須滿足 $X_{new} \in C_{free}$ 且 $Dis(X_{near}, X_{new}) = \varepsilon$ 的條件，其中 $\varepsilon > 0$，為 RRT 成長的最小單位長度(惟一的例外是當 $Y_{rand} \in T_k$ 時 $Y_{rand} = X_{near} = X_{new}$ 會成立)。若是 X_{new} 存在，則 T_k 便會增加一個新的節點。令 T_{k+1} 代表新的 RRT，則 $T_{k+1} = \{X\} + X_{new}$。欲建構一顆具有 K 個節點的 RRT 之演算法如下：

```
Build_RRT (X_root)
1.  T.init (X_root);
2.  for k =1 to K do
3.      Y_rand ← Random_Configuration ( );
4.      Extend (T, Y_rand);
5.  return T;
```

```
Extend (T, Y_rand)
1.  X_near ← Nearest_Neighbor (T, Y_rand);
2.  X_new ← New_Configuration (X_near);
3.  if X_new is NOT Null
4.      T.add_node (X_new);
5.      if X_new = Y_rand
6.          return Reached;
7.      else
```

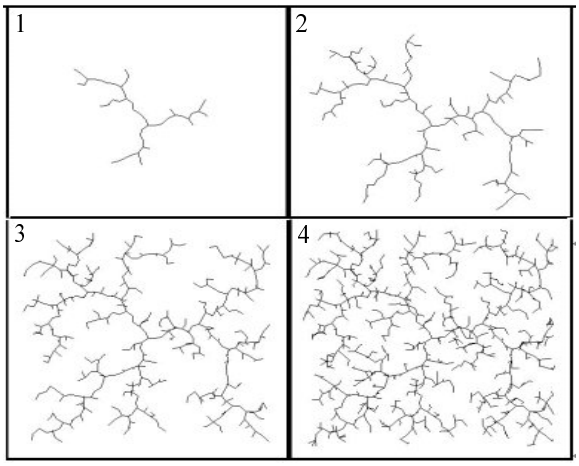

圖 2: 三維空間中 RRT 成長情形之投影圖，其中 RRT 之節點數依序為 50、200、500 以及 1000，整個組態空間之大小為 128x128x100。

8. **return** *Advanced*;
9. **return** *Trapped*;

其中 Extend 這個函式有三個可能的回傳值，*Trapped* 代表沒有 X_{new} 的存在，*Advanced* 代表成功尋找到 X_{new}，但 $X_{new} \neq Y_{rand}$，而 *Reached* 則是代表不但能求得 X_{new}，並且 X_{new} 即等於 Y_{rand}，這些回傳值的意義將會在下節使用 RRT-Connect 模組時實際產生作用。而圖 2 則是 RRT 在沒有障礙物之環境下的成長情形，由圖中可以觀察到 RRT 的重要特性：RRT 的分枝具有往空曠的地區成長的趨勢。這個特性使得 RRT 的節點可以快速的分佈於整個環境，輕易地探索未知的區域，因此即使是高維度空間中的運動計畫，RRT 亦能夠勝任。

3.2 RRT-Connect

RRT-Connect[9]被設計用來連接兩顆 RRTs，其實際上即為一個單一查詢的路徑計畫器(single-query path planner)。假設給定起點以及終點的組態為 X_{init} 和 X_{goal}，則利用 RRT-Connect 的演算法可以求得一條由 X_{init} 到 X_{goal} 的連續路徑，或者當 X_{init} 與 X_{goal} 之間沒有適當的路徑時則會告知搜尋路徑失敗，其演算法如下：

Connect (T, Y)
1. **repeat**
2. $S \leftarrow$ Extend (T, Y);
3. **until** (S $\neq$ *Advanced*)
4. **return** S;

RRT_Connect_Planner (X_{init}, X_{goal})
1. T_a.init (X_{init})；T_b.init (X_{goal});
2. **for** k =1 to K **do**
3. $Y_{rand} \leftarrow$ Random_Configuration ( );
4. **if** (Entend (T_a, Y_{rand}) $\neq$ *Trapped*)
5. **if** (Connect (T_b, T_a, X_{new}) = *Reached*)
6. **return** Path(T_a, T_b);
7. Swap(T_a, T_b);
8. **return** *Failure*;

其中 Connect 為一個 gready heuristic 函式，當給定 RRT T 以及組態 Y 時，Connect 能讓 T 不斷地朝 Y 生長，直到 T 包含 Y 或者是遇到障礙物而回傳 *Trapped* 為止。其主要的想法是類似於傳統的雙向搜尋技術(bidirectional search)[9]，分別以起點和終點為樹根形成兩顆 RRTs，獨立地搜索各自周圍的環境，直到兩顆 RRTs 有交集時即可求得由起點到終點的路徑。這樣的演算法具備機率上的完整性(probabilistic completeness)，因為只要空間區域上是相互連接的(connected)，RRT 的分枝就有機會能夠延伸到達。

3.3 整合 learning phase 與 query phase

使用 Roadmap 的方法來解決運動計畫的問題可以分為兩個 phase：learning phase 跟 query phase。learning phase 主要用來建構整個自由空間中的街圖，以供之後的 query phase 來使用；而一個街圖的完整性，則可以依據 query phase 下所做查詢的成功機率來度量。一般來說，街圖可以用幾種形態具體表現出來：graph、bitmap、tree 或者是 forest 等等。這些資料結構所構成的街圖都有一個共同點：其主要的成份是在自由空間中之組態的部份集合，而根據集合的大小，可以初步地估計一份街圖的完整性。然而在高維度的空間裡，要建構一份具有完整性的街圖，往往其所花費的時間以及儲存整個街圖的資料結構所需的記憶體都是非常龐大的。也因此，利用取樣的方式來建構一份街圖變成較為可行的方法，而取樣數目的多寡，通常會與 query phase 下的成功機率成正比；同時，建構街圖所花費的時間亦會隨著增加。

當運動計畫的工作空間較小時，通常會先在 learning phase 下建構一完整的街圖，之後便停留在 query phase 下不需要再做改變。假若工作空間太大，建構街圖所需的時間太長，或者是要維護一份完整的街圖所需的記憶體太大時，通常就必須以取樣的方式，先建構一份主要的街圖來供 query phase 下查詢使用；一旦查詢失敗時，便會再次進入 learning phase 下，花費一段事先定好的時間來將目前的街圖再做擴充，如此不斷地再兩種狀態下做切換，直到 query phase 下的查詢成功為止[13]。然而這樣的作法，在查詢失敗的情況發生時，可能必須進入 learning phase 一次以上，而所花費的時間也變得無法保證，這會在實際的應用上造成限制。因此，我們設計將 learning phase 以及 query phase 整合在一起，當 query phase 下查詢失敗時，我們使用 RRT-Connect 的演算法來進

行單一查詢的運動計畫，使得在可預期的時間內可以得到該次查詢所要的結果，然後再將計算過程中新產生的 RRTs 加入原來的街圖之中，我們稱此街圖為 RRT-Roadmap，並可將其用於往後的查詢之中。

3.4 RRT-Roadmap

RRT-Roadmap 本質上是由 RRTs 所構成的 forest，其中的 RRTs 是逐漸地加進來，而非在同一時間所構成，並且經由不斷地作 query 的動作，所有在 forest 中的 RRTs 會有慢慢合而為一的趨勢。欲維護一個 RRT-Roadmap，最基本的步驟就是將兩顆 RRTs 合併(merge)在一起，因為在同一顆 RRT 中的所有節點都是互相可連結的(connected)，所以當給定的起點以及終點可以同時連接上同一顆 RRT，則可以很容易地找到一條由起點到終點的路徑。然而，在許多的情況下，起點跟終點並非一開始就能夠連結上同一顆 RRT，並且也有可能在連結不上任何一顆 RRT 的情況下，必須以起點或是終點獨立建構出新的 RRTs，因此，往往需要透過與其它的 RRTs 做合併的動作，最後才能使起點以及終點連結在一起。

欲合併兩顆 RRTs，基本上也是利用 RRT-Connect 演算法來達成。在圖 3 中原本有兩顆 RRTs，T_A 及 T_B。選定了 T_A 中的一個節點 X_A 做為 T_B 進行 Connect 操作之參考點，如此可讓 T_B 延著 X_A 的方向持續成長(圖 3 中白色的圓圈即代表 T_B 新增加之節點)，當 T_B 包含了 X_A 時，便開始往回做顛倒每個節點的 parent-child 關係之動作，直到到達 T_B 的樹根為止，則 T_A 和 T_B 到此即合為一顆 RRT。以下列出其演算法，其中 Merge_RRTs 的參數 X_A 是代表 T_A 中用來與其他的 RRTs 做合併的節點，是由我們所給定而非代表任何特別的節點。此外在演算法中出現一全域變數 *Forest*，其儲存了目前在 RRT-Roadmap 中所有的 RRTs：

```
Merge_RRTs(T_A , X_A)
1.  for each T in Forest
2.      if (T ≠ T_A)
3.          if (Connect (T , X_A) = Reached)
4.              Reverse_Parent(T , X_A);
5.              Forest.remove(T);
6.              return True;
7.  return False;
```

有了 Merge_RRTs 這個函式後，即可以用它來完成一個 RRT-Roadmap 的運動計畫器，其演算法如下。詳細說明列於演算法之後。

```
RRT_Roadmap_Planner(X_init , X_goal)
1.  T_init.init(X_init); T_goal.init(X_goal);
```

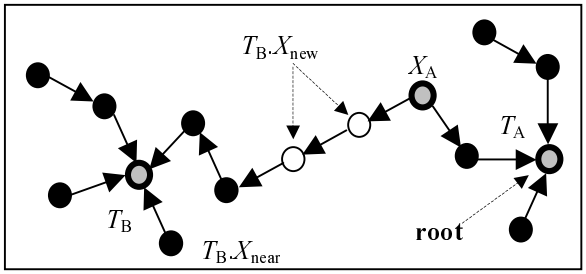

圖 3: 兩顆 RRTs 做 merge 的情況，TB 根據 XA 做 Connect 的操作（箭頭為指向 parent 的指標），直到 TB 與 TA 在 XA 交集時，TB 由 XA 開始作 Reverse_Parent 的操作，直到 TB 之樹根為止。

```
2.  Forest.add(T_init); Forest.add(T_goal);
3.  Merge_RRTs(T_goal , T_goal. X_root);
4.  if (Merge_RRTs(T_init , T_init. X_root))
5.      if (T_init.tree_id = T_goal.tree_id)
6.          return Path(T_init. X_root , T_goal. X_root);
7.  for k =1 to K do
8.      Y_rand ← Random_Configuration ( );
9.      if (Entend (T_init , Y_rand) ≠ Trapped)
10.         if (Merge_RRTs (T_init , T_init. X_new))
11.             if (T_init.tree_id = T_goal.tree_id)
12.                 return Path(T_init. X_root , T_goal. X_root);
13.     Swap(T_init , T_goal);
14. return Failure;
```

在演算法第 3 至 6 行是屬於 query phase 的部份。以 X_{init} 及 X_{goal} 為樹根所建構的 RRTs 先試著與 *Forest* 中其它的 RRTs 做連結。若是成功的話，則檢查 X_{init} 與 X_{goal} 是否屬於同一顆 RRT，如果結果是肯定的話，就可以直接回傳兩者間的路徑；假使兩者仍分屬於不同的 RRTs(代表尚無路徑)，則進入到 7 至 13 行的單一查詢式運動計畫的部份，迴圈中每次執行都會隨機地在自由空間中選取一個組態，並以此組態來促進 T_{init} 或 T_{goal} 的成長，使其更有機會與其它的 RRTs 做合併。這樣的設計，一方面使得現有的 RRT-Roadmap 可以被單一查詢的運動計畫所利用而更快速求得解答；另一方面，計算後的結果又可新增到 RRT-Roadmap 之中，使整個街圖更加的完備，可說是相輔相成。

4. 在動態的環境下修改 RRT-Roadmap

利用建構街圖的方法來解決運動計畫的問題，其所遭遇到的最大限制便是無法應用於動態的環境之中。假如工作空間中的障礙物會隨著時間而改變位置，那麼先前所建構好的街圖就有可能被破壞。然而重新建立整個街圖是很不經濟的，尤其是當環境經常會變動時；因此，必須有一能夠動態地修正部份街圖之機制，使其可以重新適用於新的狀況。

欲修正一份街圖，最直覺的方法就是把變為不合法的組態拿掉，然而並非所有的資料結構都可以輕易地辦到這點。但是就 RRT-Roadmap 而言，其不過是將擁有不合法節點的 trees 先修剪成許多的 sub-trees，然後捨棄掉所有不合法的 sub-trees，並保留下其它合法的部份。對於以 forest 的形式來組織而成的 RRT-Roadmap 來說，其本質完全沒有改變。而要達到這樣目標的基本作法，主要可以分成下面四個步驟。

Step 1：在障礙物移動之後，先計算障礙物的邊界(*bounding box*)，並將 RRT-Roadmap 裡，落在該區域中的所有節點先篩選出來。
Step 2：運用碰撞偵測的演算法，將 step1 所篩選出來的節點再次過濾，並將不合法的節點存到叫做 *Illegal_Nodes* 的 queue 中。
Step 3：由 *Illegal_Nodes* 中取出一個節點 X_{illegal}，然後檢查 X_{illegal} 的所有子節點，看其是否也存在於 *Illegal_Nodes*。若是其不在 *Illegal_Nodes* 中，代表該子節點為一合法的節點，並將其儲存於叫做 *Candidate_Roots* 的 queue 中，待檢查完 X_{illegal} 所有的子節點之後，將 X_{illegal} 由 RRT-Roadmap 中移除，並重複此步驟，直到 *Illegal_Nodes* 中不再有任何節點為止。
Step 4：由 *Candidate_Roots* 中取出一個節點 X_{root}，然後以 X_{root} 為樹根建立一顆新的 RRT，並將其加入 RRT-Roadmap 之中，然後重複此步驟，直到 *Candidate_Roots* 中不再有任何節點為止。

在上面幾個步驟之中，最花費時間的是步驟 2，因為其必須要做即時的碰撞偵測運算。即使是我們在步驟 1 時就先過濾掉大量不必要做碰撞偵測檢查的節點，但是當障礙物本身所佔有的區域較大時，仍舊是要花費可觀的時間。然而情況也並不是那麼糟糕，因為在實際上應用時，我們甚至可以將整個步驟 2 給省略掉，並直接將步驟一中所篩選出來的節點直接存到 *Illegal_Nodes* 當中即可，理由有幾點：

1. 我們的目的只是為了要移除掉 RRT-Roadmap 中所有不合法的節點，而步驟 1 中，利用障礙物的 bounding box 所篩選出來的節點皆滿足不合法節點的必要條件。
2. 移除掉較多的節點數，對於之後的運用並不見得會比較不好，甚至可能會更有幫助，因為假使障礙物是做連續性的運動，或者是在固定的區間中變換位置，那麼預先移除掉其周圍的節點，可以避免掉往後多餘的運算。
3. RRTs 的特性是會往較空曠的地區擴張成長，所以即使部份區域的節點暫時被移除，必要時，RRTs 的枝葉亦能快速地延伸至此，不需要擔心會對往後的路徑計畫造成太大的影響。

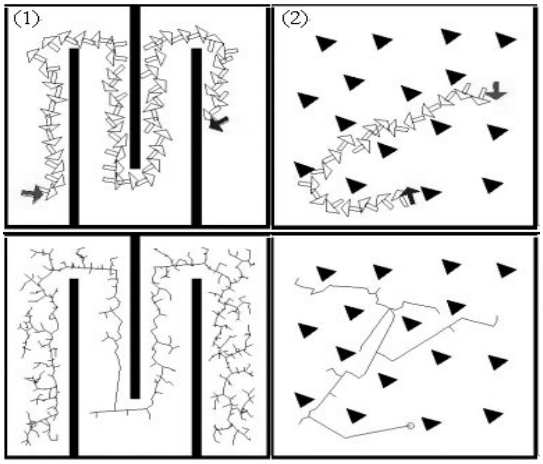

圖 4: 路徑計畫器的執行結果比較。(1)節點總數為 699，隨機取樣總數為 810 次，計畫時間約 0.4 秒；(2)節點總數為 112，隨機取樣總數為 28 次，計畫時間約 0.1 秒。

5. 實驗

利用前兩個小節的演算法，我們實作了一個二維工作空間的路徑計畫器，所使用的語言為 Java，程式執行的平台為 Pentium II 350。在例子中以箭頭的圖案來代表 *robot* 的起點與終點，而 *robot* 在此空間裡會有三個自由度，以 (x, y, θ) 表示其組態，而組態空間的大小為 128x128x100。

5.1 靜態場景實驗

圖 4 為路徑計畫器執行結果的擷取畫面，其中展示了兩個例子，在第一個例子的場景中，進行第一次路徑計畫所需的時間會比較長，因為以起點和終點為樹根的兩顆 RRTs 比較不容易碰在一起，但是當第一次的計畫完成之後，RRT-Roadmap 已分佈於整個場景，這對於後續的路徑計畫而言，就只是單純的做查詢，而不需要再做隨機取樣的動作，因此計畫的平均時間也會降到 0.1 秒以內。而第二個例子中，RRT-Roadmap 的成長情形會比較緩和，圖 5 為其隨機取樣次數的統計，我們以亂數的方式來決定 *robot* 的起點和終點，並連續進行兩百次的路徑計畫，來觀察其隨機取樣次數增加的情況。圖中曲線的水平部份代表路徑計畫器處於 query phase 的狀態，也就是只需要對 RRT-Roadmap 做查詢即可求得路徑，而不用再以隨機取樣來增加節點數，並且隨著計畫次數的增加，所需做隨機取樣次數的比例也會明顯地減少。

理論上，當 RRT-Roadmap 成長到某一個程度時，運動計畫就會保持在 query phase 的狀態下，而隨機取樣的次數也不會再增加。但就我們實驗的結果顯示並非如此，原因在於以亂數來決定 *robot* 的起點跟終點時，假使其中之一剛好鄰接著障礙物，則可能會發生類似 local minima 的情況，

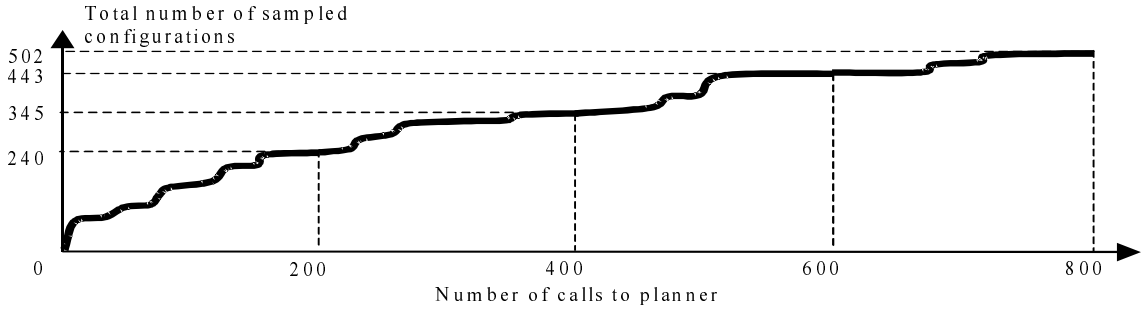

圖 5: 計畫器學習過程中的取樣點數。縱軸為隨機取樣總數，橫軸為計畫器被呼叫的總次數。所使用的例子為圖 4.2，隨著計畫次數的增加，隨機取樣的成長比例會逐漸趨緩。

此時 *robot* 欲擺脫障礙物，只能就幾個固定的方向移動，因此不容易連接上現有的 RRT-Roadmap，惟有再增加隨機取樣的數目，迫使 RRT-Roadmap 再成長，才能有機會脫離障礙物的邊緣。

5.2 動態場景實驗

在第 4 節所提出的動態修改 RRT-Roadmap 之演算法，我們將其整合到我們的路徑計畫器中，圖 6 即為我們所作實驗的說明。圖 6.1 是場景的擷取圖，為了清楚呈現修改 RRT-Roadmap 的過程，我們首先以亂數產生 *robot* 的起點與終點，並連續做 1000 次的路徑計畫，使得 RRT-Roadmap 佈滿整個場景，如圖 6.2 所示。接下來我們移動右下角的障礙物到場景的中央，這個動作會觸發路徑計畫器對 RRT-Roadmap 做更新的動作，結果會有近 300 個節點從 RRT-Roadmap 中被移除，並且產生 22 顆新的 RRTs。圖 6.3 中的圓點代表每一顆 RRTs 的樹根，新產生的樹根會圍繞著變動位置之障礙物的邊緣，而整個過程大約花費了 0.05 秒左右。第四張圖是再經過 500 次亂數計畫路徑後的結果，原本因為障礙物移開所留下的空白區域，最後會逐漸再被 RRT-Roadmap 所填滿。

5.3 問題討論

在圖 7 中我們實驗了一個較為困難的例子，可以讓我們較容易觀察出 RRT-Roadmap 演進的情形。在該場景中有許多的狹窄通道，僅剛好可以讓箭頭形狀的 *robot* 在特定的角度下通過，因此往往需要許多次的隨機取樣後才能讓 RRT-Roadmap 覆蓋到這些關鍵的地點。不過正由於整個場景被障礙物切割為幾個主要的部份，所以 RRT-Roadmap 的 forest 結構可以很清楚地被呈現出來。在圖 7.1 中可以看出 RRT-Roadmap 包含了六顆 RRTs，並隨著路徑計畫次數的增加，除了左上角，其餘的部份會合併為一顆 RRT，此時的 RRT-Roadmap 包含了兩顆 RRTs，共 2359 個節點。如果我們持續增加隨機取樣的次數，最後 RRT-Roadmap 將會成長為圖 7.4 的情況。

圖 7 這個例子中有一點值得注意的，就是左上角的區域和其它的部份是沒辦法連結在一起的，因為箭頭形狀的 *robot* 無法通過該區域的通道，所以如果 *robot* 的起點或終點之一落於此區域，則路徑計畫的結果將會失敗，而其所造成的結果是 RRT-Roadmap 中的節點數會大量的增加，並且有許多的節點實際上是不必要存在的。這些多餘的節點除了會浪費記憶體的空間外，亦會影響路徑計畫時的效率。尤其對必須長時間執行的程式而言，RRT-Roadmap 無限制的成長必定會造成大問題。解決的方法之一是做場景的分析，找出互相不可連結的區域，以避免進行不必要的路徑計畫程序，或者是在街圖建立的過程中，隨時對 RRT-Roadmap 進行去蕪存菁的工作，而到目前為止，我們並未對 RRT-Roadmap 節點的成長情形作控管，這方面的工作我們將留待以後補充。

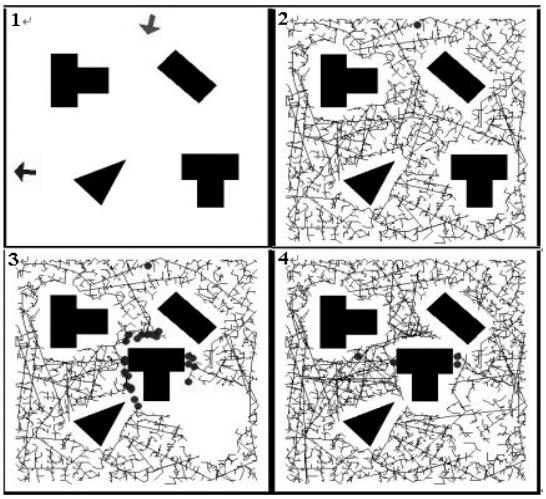

圖 6: 動態場景之實驗。(2) 1000 次計畫所產生的 RRT-Roadmap，節點總數為 2962；(3)移動右下角障礙物至中央，動態更新的結果產生 22 顆新的 RRTs，所需時間約 0.05 秒；(4)再經過 500 次的亂數路徑計畫，RRTs 合併為 4 顆，節點總數為 3987。

6. 結論

在本文中我們改良 RRT 的資料結構，建構出另

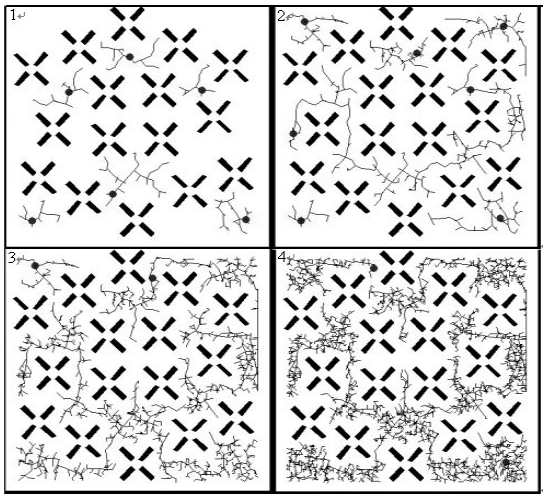

圖 7: RRT-Roadmap 的成長過程，圖中的圓圈代表 RRTs 的樹根，而圖的左上角區域與其他的區域是分離的(robot 無法通過)。

一種資料結構 RRT-Roadmap。這種資料結構是以 RRTs 所組織而成的 forest，透過整合傳統的單一查詢跟多次查詢運動計畫之概念所建構而成的漸進式街圖。除了不需要事前的處理程序即可運作外，亦具備學習和記憶的功能，並且在動態的環境之下，RRT-Roadmap 也能有效地作即時的修正。利用在本文中提到的演算法，我們實作了一個路徑計畫器，並經由觀察幾個範例的進行過程，可以讓讀者更深入的了解 RRT-Roadmap 的特性，並且由實驗結果的分析，我們提出目前的一些問題所在，而其中最關鍵的問題為必須控制 RRT-Roadmap 中節點數的成長；除了如何增加新的節點是較為有效以及經濟的方式之外，亦必須有一個機制可以事後來對 RRT-Roadmap 做整理和修剪的工作，這牽涉到如何來評估一個節點在 RRT-Roadmap 中的重要性，以及如何在有限的時間裡頭來完成這項工作，而這部份將會是我們未來研究的方向之一。

誌謝

本文感謝國科會研究計畫 (NSC 89-2218-E-004-009) 經費贊助，特此誌謝。

參考文獻

[1] J. Barraquand, L. Kavraki, J.C. Latombe, T.Y. Li, and P. Raghavan, "A Random Sampling Scheme for Path Planning," in *International Journal of Robotics Research*, 16(6), P759-774, December, 1997.

[2] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for find-path with rotation," *IEEE Transaction on System, Man, and Cybernetics*, vol. 15, pp. 224-244, 1985.

[3] L. Kavraki, P.Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces," *IEEE Transaction on Robotics and Automation*, 12:566-580, 1996.

[4] P. Khosla and R. Volpe, "Superquadric artificial potentials for obstacle avoidance and approach," in *Proceedings of IEEE International Conference on Robotics and Automation, Philadelphia*, pp. 1178-1184, 1988.

[5] D. E. Koditschek, "Robot planning and control via potential functions," in *The Robotics Review 1, Cambridge: MIT Press*, O. Khatib, J. J. Craig, and T. Lozano-Perez, editor, pp. 349-367, 1989.

[6] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," *in Proceedings of the International Conference on Robotics and Automation*, vol. 2, pp. 995-1001, 2000.

[7] J. Latombe, *Robot Motion Planning*, Kluwer, Boston, MA, 1991.

[8] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Iowa State University*, 1998.

[9] S. M. LaValle and J. J. Kuffner. "Randomized kinodynamic planning," in *Proceedings of 1999 IEEE International Conference on Robotics and Automation*, 1999.

[10] T.Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu, "Automatically Generating Virtual Guided Tours," in *Proceedings of the Computer Animation '99 Conference*, Geneva, Switzerland, pp. 99-106, May 1999.

[11] T.-Y. Li and T.-H. Yu, "Planning Object Tracking Motions," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, May 1999.

[12] P. C. Nelson and A. A. Toptsis, "Unidirectional and bidirectional search algorithms," in *IEEE Software*, vol. 9, no. 2, pp. 77-83, 1992.

[13] M. H. Overmars and P. Svestka, "A Probabilistic Learning Approach to Motion Planning," in *Algorithmic Foundations of Robotics*, K. Goldberg, D. Halperin, J. C. Latombe, R. Wilson, editor, pp. 19-38, 1994.

[14] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem – II: General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, pp. 298-351, 1983.