虛擬環境中第三人稱化身之智慧型控制

許書瑋
國立政治大學資訊科學系
台北市指南路二段 64 號
(02)29387170
g9204@cs.nccu.edu.tw

李蔡彥
國立政治大學資訊科學系
台北市指南路二段 64 號
(02)29387170
li@nccu.edu.tw

摘要

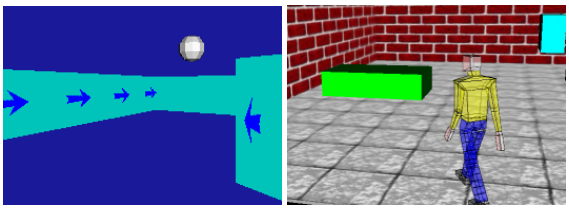
在 3D 虛擬環境中，使用者透過對化身的控制進行場景瀏覽等各種活動。所謂智慧型化身之控制，是指在化身控制系統中，融入一些人工智慧的方法，協助使用者更有效率的控制化身的運動。本論文提議將智慧型控制的概念延伸到第三人稱的控制方式上。在第三人稱的化身控制上，我們以「組態-時間」街圖的新觀念設計運動計畫演算法，並加上讓運動計畫與控制播放兩個程序協調並行的機制，使系統能在符合即時性的效能下，自動為化身搭配能夠適時閃避障礙物的上半身肢體運動。我們相信此類智慧型控制介面，可增加化身在虛擬環境中控制的便利性，並提高化身在動作呈現上的擬真度。

關鍵詞：即時運動計畫、智慧型化身控制、虛擬環境、電腦動畫

一、 簡介

近年來，發展快速的電腦 3D 圖學技術，除了代表電腦硬體技術日新月異之外，也意味使用者對電腦 3D 應用的需求與期望。互動式虛擬環境就是其中一種被廣泛應用的 3D 圖學技術。虛擬實境是由電腦模擬出來的互動式三度空間，讓使用者能夠化身為虛擬人物，進入其中體驗擬真的虛擬世界，與世界中的事物產生互動。一般人對虛擬環境的印象，可能是需要穿戴一些特殊設備，如虛擬頭盔、資料手套，然後才能遨遊於虛擬世界中。這種是屬於沈浸式(immersive)的虛擬環境。另一種在個人電腦上較為普及的，屬於非沈浸式(non-immersive)的虛擬環境，只需透過一般的電腦螢幕視窗加上滑鼠鍵盤等輸入裝置，即可進入虛擬世界中。

通常我們稱使用者在虛擬環境中所操控的虛擬人物為「化身」(avatar)。常見的化身型態分為第一人稱與第三人稱兩類，如圖一所示。在第一人稱化身的設計上，使用者操控的是化身視點(viewpoint)，隨著視點的移動，所呈現出來的畫面就如同真的在虛擬環境中移動一般，不過無法看見化身本身的模型。對於第三人稱的化身，操作的則是人形化身的模型(humanoid model)。在操控過程中，除了場景的變化之外，也能夠看見化身的模型。當化身本身的運動對虛擬環境的使用者有意義時，像是 3D 遊戲或是虛擬工作訓練這類型的應用，就需要使用這類第三人稱的化身。



(a) 第一人稱化身(視點)　　(b) 第三人稱化身
圖 一：兩種不同類型化身控制

在過去文獻中，有一些研究使用了「運動計畫」的方式協助使用者在虛擬環境中進行瀏覽[6][13]，其中[6]這個研究是替使用者產生了整體導覽的路徑，讓使用者直接順著這些路徑進行瀏覽；[13]則是由使用者自由的在虛擬場景中瀏覽，當視角卡住時，才啟動路徑規劃協助使用者脫困。[5][14]則是提出使用「虛擬力場」來協助使用者在虛擬環境中瀏覽。力場的概念是在障礙物周圍部署排斥力，當化身靠近障礙物時將他推開，減少碰撞到障礙物的機會。這些研究也都被證實能夠提升使用者在虛擬場景中瀏覽的效率。不過目前大多數的智慧型化身控制系統，都是針對第一人稱化身設計的，並沒有為第三人稱化身特別設計的智慧型化身控制系統。

本論文的目的是要將智慧型的控制系統延伸到第三人稱化身上。第三人稱化身的控制問題，比第一人稱的控制複雜。舉例來說，如果控制的是第一人稱視點，遇到之前提到的視點碰撞問題，只要能夠從碰撞的地方將視點帶開，就算是解決問題了。但是對於第三人稱化身，還必須額外考量化身肢體運動上的因素。例如化身在前進過程中，發現手臂快要和障礙物發生碰撞了，手部運動的軌跡應可稍微修正即可通過，如此更符合真人的反應。但目前在遊戲中或是各類型的虛擬環境的作法，為了運算效能，對於第三人稱的局部肢體的運動，通常是套用預先準備好的固定動作，因此無法達到此目標。以計畫的方法來產生化

身的動作固然可能，但在計算的複雜度上通常難以達到即時性的效能。在動作真實性與時間效能兩者之間，往往不容易達到兩全其美的目標。

我們的想法是，把用一些常用的局部運動動畫轉化為化身活動能力圖。當使用者在操控化身的過程中，將化身活動能力圖向未來時間軸延伸成「組態-時間」空間街圖(Configuration-Time Space Roadmap)，在此街圖上，搜尋適當的局部運動。如此一來，當化身被控制的過程中，在沒有障礙物的情況下，能夠自然的行走；在遇到障礙物時，當軀幹無法通過就繞過障礙物，否則就直接調整肢體動作避開碰撞。我們認為這樣的智慧型化身控制系統，在控制上的便利性與呈現上的真實性都能兼顧。

二、 相關研究

在這一章主要是回顧與本研究相關的研究成果與文獻。第一個部分介紹一些與虛擬環境中化身控制相關的研究，特別是一些創新的智慧型控制輔助介面；第二部分介紹運動計畫演算法；最後一部份我們將探討一些和人體動畫有關的研究。

(一) 虛擬環境中智慧型 3D 操控介面

3D 操控介面，是使用者和虛擬環境溝通的管道。在 3D 操控介面的設計方面，因應用的不同而衍生出許多不同的類型的設計，特別是對沈浸式的虛擬環境。對非沈浸式的虛擬環境，早期對操控介面的研究，大多是針對如何在較低維度的輸入裝置與虛擬物體的控制間，建立一個直覺的、容易理解的對應；例如如何使用 2D 滑鼠來進行 3 維空間中一些操作問題[1][9][10]。漸漸的，有些學者認為可以透過人工智慧的方式，來協助使用者對虛擬環境中事物的控制，因此陸續有一些協助的智慧型介面被提出來。例如，引導式的協助瀏覽機制[6]，或是加入路徑規劃器、虛擬力場等人工智慧技術的輔助瀏覽介面[3][11][13][14]。這些研究的目的，其實都是為了讓使用者在虛擬環境中，更容易的操作。不過目前這些機制，大多都是用在第一人稱化身身上，也就是協助使用者操控視點在虛擬環境中進行瀏覽。[3]這個研究雖然把運動計畫的輔助機制應用到第三人稱的化身身上，但是並未為化身搭配任何的肢體運動，像是走路時的手臂擺動、步伐的改變等等。

(二) 運動計畫演算法

運動計畫的技術起源於機器人學的領域，主要目的是替機器人或機器手臂，從我們指定的起始狀態到目標狀態，規劃出一條不會與環境中障礙物碰撞的路徑，使得機器人或機器手臂可以順利的移動。除了機器人自動化之外，近年來運動計畫所發展出來的技術也被廣泛的應用到其他領域上，如電腦動畫[16]，或是 3D 虛擬實境中的人機介面操作輔助[3][13]等。

在運動計畫的技術當中，我們通常稱要被規劃的物體為機器人(robot)，而機器人所處的工作環境我們稱工作空間(Work Space)。在工作空間中，機器人的狀態是用組態(configuration)來描述，而全部組態的集合稱組態空間(Configuration Space)。在組態空間之中，和障礙物(obstacle)沒有碰撞的子空間集合，稱為自由空間(Free Space)。舉例來說，X 是一個能夠在 2 維平面上移動的機器人，所以 X 的工作空間為一個 2 維空間。而 X 的組態空間維度，要看 X 在這個工作空間上的自由度而定。假設 X 是單一個剛體，可以移動、轉動，則描述 X 的組態形式為 $X=(x, y, \theta)$，我們說 X 組態空間的維度是 3 維。

街圖法是文獻中目前較受歡迎的方法。以街圖法的方式解決運動計畫的問題，可分為兩個階段：前處理階段(Preprocess)及查詢階段(Query)。在前處理階段，我們先建立自由空間中的街圖連結，以供查詢階段使用。在查詢階段時，如果能夠將起始組態和目標組態連接上現有的街圖，就可以得到一條路徑。街圖法的好處，就是一旦街圖建立完成，就可以重複查詢使用。街圖產生的方法有很多種，其中快速隨機拓展樹[12] (Rapidly-exploring random tree，RRT)可以算是基本隨機式街圖的一種變形。RRT 能夠從一個特定的組態出發，快速向四周拓展。和一般街圖法不同，RRT 並非直接當成街圖使用，而是搭配 RRT-Connection 演算法[7]，從起始和目標組態各長出一棵 RRT，再讓這兩顆 RRT 連結起來，完成路徑的搜尋。

(三) 人體動畫

當我們在虛擬環境中選擇使用第三人稱人形化身時，人體動畫(humanoid animation)就成了必需考量的部分。人體動畫由於本身的特性，經常被分為兩部分來看待，全域運動(global motion)與局部運動(local motion)。全域運動指的是路徑上的變化，局部運動則是肢體動作的變化。不論是全域或局部的運動，最基本的產生方法就是由動畫師編修關鍵格(Key Frame)，或是透過運動擷取(Motion Capture)這兩種方式產生。這兩種方法共同的缺點是動畫產生後就固定了，所以比較適合影片、卡通這種一次性的應用。對於 3D 遊戲、虛擬環境這一類互動的應用裡，虛擬角色的體型、場景、移動的路線可能都是會變化的，虛擬角色使用的全域或局部運動可能隨時都需要做些許的調整。為了使人體動畫能夠比較彈性的被產生或被重用，有許多研究被提出。研究策略上大概可分為兩類，分別是樣本資料驅動(Data Driven)與程序性方式(Procedure)。

樣本資料驅動的方法類似動畫資料已經存在了，然後考慮外在的變因，去調整動畫資料，讓動畫資料可以符合外在變因的要求。外在變因可能是環境改變

了，例如在[2]中，將動畫中的關鍵格經過變形調整後，讓原本走路的動作，變成跨過地面上有凸起物體的動畫。變因也可能是動作本身，例如讓運動擷取器擷取下來的動作改變運動的風格[4]。樣本資料驅動方法的好處是，動作真實性高，但缺點就是實際上能夠變化的幅度沒辦法太大。

以程序性方法為基礎這一類的研究並不像樣本資料驅動先有一些動畫資料存在了，而是由電腦程式自行推理來產生動畫。在產生時，可以同時把外在的變因納入考慮，例如環境、運動的路線等，所以能彈性的根據變因產生出合適的動畫。產生的法則，或者說是模擬的法則，可以是用運動計畫的方法或是物理上的機構學或動力學等，來推算出角色該如何動起來。例如[16]這個兩個研究，都使用運動計畫的方法，分別自動產生虛擬人物手臂操作物體的動畫。[15]這個研究能夠在具有高低起伏的 3D 場景中，自動產生合適的下半身行走動畫。程序性方法的好處是動作的產生上可以有很高的彈性，缺點則是在動作上的擬真程度仍不及資料樣本驅動的方式。

上面這些研究，對於人體動作產生上的彈性與重用性都有貢獻，但是絕大部分都是針對非即時動畫方面的應用，鮮少是能夠在即時虛擬環境系統中應用的，只有[15]能夠達到即時性的應用。因此如何即時的產生具真實性又同時能夠根據環境調整的化身局部肢體動作，是本研究最大的挑戰之一。

三、 系統概觀

(一) 問題描述

在使用者輸入控制指令後，首先要解決的問題是全域路徑的計畫，必須先為使用者決定一個短程的目標，以及規劃出一段往短程目標前進的路徑。有了全域路徑之後，第二個問題是，我們必須為這段路徑，搭配上半身的局部肢體運動。局部運動有兩個特色，第一，這些動作大多是下意識的，屬於次要運動(secondary motion)的一種；第二，這些運動都符合週期性的運動樣式(cyclic motion pattern)，行走中會反覆的進行。因此局部肢體運動的產生有兩個限制條件，首先是肢體運動必須盡量保持走路時上半身的運動樣式，包含了雙手自然擺動與保持腰部直立，這個限制屬於非嚴格限制(soft constraint)。第二個限制則是屬於嚴格限制(hard constraint)，即在搭配動作的過程中，肢體的運動軌跡，不能夠與障礙物發生任何碰撞。

圖 2 為化身自由度模型的架構圖。我們以 R 代表系統中的化身。R 上半身的部分共有 13 個自由度。以 C 代表 R 的組態空間，表示 R 在進行運動計畫時的參數空間，或稱組態空間(Configuration Space)。我們沒有直接使用與自由度空間相同的 13 維空間當成 R 的組

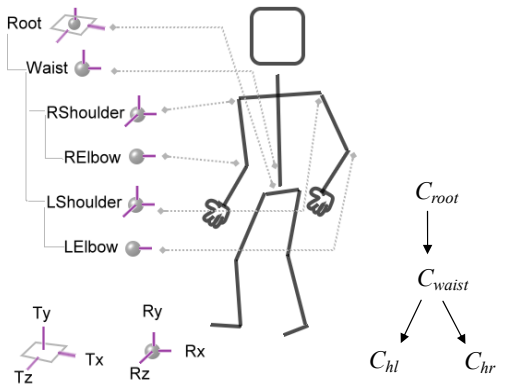


圖 2：化身上半身自由度模型，T：位移，R：旋轉

態空間，而是根據人體架構的特性，將 13 個維度分成四個群組(如圖 2)，以 J 來表示四個群組的集合，J ={ $root$, $waist$, hl, hr }，故 C=(C_{root}, C_{waist}, C_{hl}, C_{hr})。每個群組有自己在計畫時的組態空間 C_j，j∈J，在 C_j 中的任一個點 q_j，稱 R 在 C_j 中的組態。q_j 是用來描述目前 R 身上，對應到 C_j 中那些自由度的值。但是因為人體架構是具有階層式架構，階層較低的 C_j，要等階層高的組態定決定後，才能夠決定。決定順序如圖 2。，另外我們定義 J^* = {$waist$, hl, hr}，J^*是四個群組中屬於局部運動的三個部分，之後在任意的定義符號 Ω_x 中， x 都是屬於 J^*。定義 J^*主要是為了後面描述方便。這種把高維度組態空間分解成幾個低維度組態空間，再依序進行計畫的方法，一般稱為分解式計畫(decoupled planning)。

(二) 基本方法

在過去一些能夠因應環境變化自動產生人體動畫的研究中[8][15]，皆能夠自動的產生讓角色閃避障礙物的動畫，但由於人體運動的計算複雜度相當高，目前這類的方法都不具有互動式虛擬環境的應用中即時性的效能需求。

由於人體運動的複雜度高，計畫人體上半身的動作將十分耗時，要能夠兼顧運動的真實性與效能的即時性將是極具挑戰性的問題。我們提議用運動計畫中街圖法的概念來加速計畫的效能。我們用圖 3 來簡單說明這個概念。首先針對不同的部位，我們將常用的肢體運動動畫、或者是可能具有避開障礙物效果的簡單肢體運動(Animation Curves)，轉化為化身該部位肢體的活動能力圖(Activity graph)。舉例來說，手臂擺動這類型的動畫就是所謂常用的肢體運動動畫；另外像是彎腰，側身，抬起手臂這類動作，就是所謂可能具有避開障礙物效果的簡單肢體運動。我們目的是要把局部運動搭配到使用者控制的全域路徑上，若以時間函式 $f(t)$來表示全域路徑，即代表我們必須把活動能力

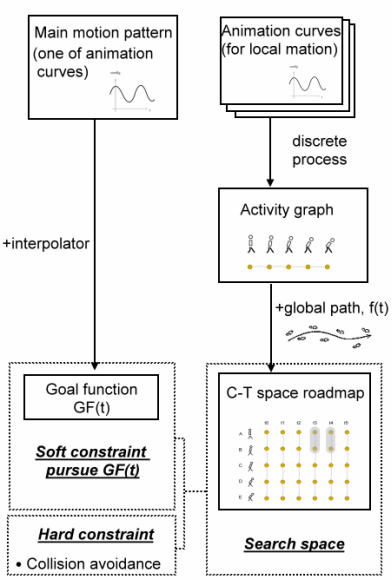


圖 3：「組態-時間」空間街圖概念

圖，向時間軸延伸成「組態-時間」空間街圖(C-T space roadmap)，再從「組態-時間」空間街圖搜尋出一段「合適」的局部肢體軌跡。所謂「合適」是指運動要有意義，而且要能夠閃避障礙物。

要如何讓搜尋出的運動有意義呢？我們可以從建立活動能力圖的動畫中，取其中之一來當成我們的主要運動樣式(main motion pattern)。例如，我們在腰部加入了彎腰，轉腰，以及保持直立這三種運動，我們可以拿保持直立這個運動曲線來當成是主要運動樣式，因為我們希望化身在移動過程中，能夠盡量保持直立的狀態；如果是手部的話，可以取手臂的來回擺動當成是主要運動樣式。主要運動樣式搭配上內插器可以轉變成目標函式(goal function)。目標函式的作用是引導搜尋，讓我們能夠搜尋出有意義的運動。我們在搜尋時必須盡量遵循目標函式的引導，除非遇到引導的動作組態會和環境障礙物發生碰撞，才能夠選擇其他的動作組態。整體來看，「組態-時間」空間街圖是搭配局部運動時的搜尋空間(search space)，而目標函式與避開障礙物則分別是搜尋時的非嚴格限制(soft constraint)與嚴格限制(hard constraint)。目前我們僅將我們的方法應用在上半身的動作搭配上，以簡化問題的複雜度。

四、 基本元件及系統運作介紹

(一) 化身活動能力圖

活動能力圖(Activity graph)在概念上，類似動作資料庫；更精確的說，應該是動作組態資料庫。我們把一些常用的動作，例如手臂擺動、彎腰等片段的動作，離散化後再重組成新的圖形資料結構(Graph)，即是所謂活動能力圖。我們希望可以在個別的 C_x 中建立一些活動能力圖，以利後續延伸為搜尋的組態空間。建構的過程如下，若 A_x 是為 R 在 x 這個群組上的肢體所預先準備的動畫，轉換的第一個步驟是把 A_x 以等距時間內插成 k 等分，得到一組活動能力組態 $Q_x=\{q_{x1},q_{x2},\cdots,q_{xk}\}$。第二個步驟要把 Q_x 中鄰近的組態相連，轉成圖形的資料結構 G_x。$G_x=(V_x,E_x)$，V_x 為 Gx 的節點集合，E_x 則是邊集合。$V_x=\{q_{x1},q_{x2}\cdots q_{xk}\}$，所有的節點就是活動能力組態，而 $E_x=\{(q_{xi},q_{xj}) \mid d(q_{xi},q_{xj}) < r\}$。$d(q_{xi},q_{xj})$為歐基理得距離(Euclidean distance)。E_x 的定義即兩兩相鄰組態距離小於 r 的所有連結。透過這兩個步驟把事先準備好的局部運動動畫 A_x 轉化成圖形的資料結構 G_x，也就是 R 在 C_x 中的活動能力圖。一個空間中也可以由多個預先定義的動畫組成活動能力圖，只要拿多組動畫切割出來的組態重新連結成一個活動能力圖即可。

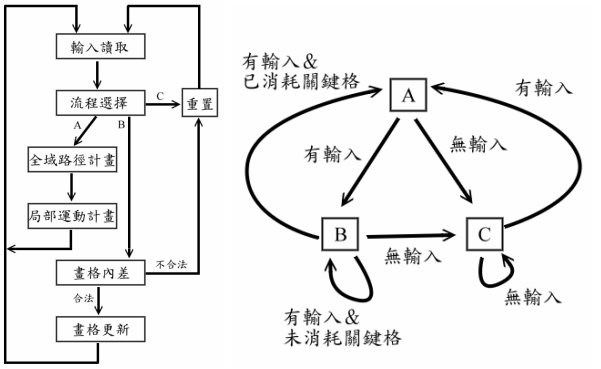
(二) 目標函式與關鍵格佇列

由於我們是以計畫的方法來為化身搭配局部動作，因此對局部肢體的組態空間 C_x 而言，在計畫的過程中，我們需要有一個目標組態 g_x，來引導肢體的移動的偏好。而此目標組態是由目標函式(goal function) GF_x 來提供。也就是 $g_x=GF_x(t)$。目標函式定義了我們在局部運動計畫過程中要跟隨的運動樣式。計畫的過程中，在任何時間都會盡量偏好目標函式提供的 g_x，除非是 g_x 是不合法的，例如會和環境障礙物發生碰撞，才會另外尋找其他的組態。

常用的目標函式種類，可以是週期型的或者是固定型的。目標函式可以直接使用預先定義的局部肢體動畫 A_x 加上對時間的內插組成，例如 $GF_x(t)$ = interpolator(A_x, t)，或是由我們自己定義。以我們的系統為例，針對 C_{waist} 所設定的目標函式是屬於固定型的，目標值固定為(0,0)，意即希望腰部的部分盡可能的不要彎曲及轉動。而對 C_{hl} 與 C_{hr} 的目標函式，則屬於週期型的，是使用預先設定好的手臂擺動動畫曲線加上內差器所組成。

在我們的系統中，為了要達成即時性的效能，必須讓運動的計畫階段與播放執行階段能夠協調並行；而要讓這兩個階段能夠協調並行，因此我們設計了一個關鍵格佇列(key frame queue)來做為運動產生與執行的緩衝區。關鍵格佇列 Q 是系統運作過程中用來儲存全域路徑與局部運動軌跡的地方。$|Q|=n$ 代表佇列的大小，而 n 為一個系統參數。

關鍵格佇列其實包含了多個子佇列，$Q=\{Q_{root}, Q_{waist}, Q_{hl}, Q_{hr}\}$。在全域路徑產生後，交給局部運動計畫搭配局部運動，如果搭配成功，會分別補充對應的關鍵格到 Q_{root}, Q_{waist}, Q_{hl}, 與 Q_{hr} 中。如果補充了 m 個關鍵格，而每一格的時間為 $T_{interval}$，則整個佇列累積的動



(a) 系統控制迴圈　　(b) 流程選擇轉換圖

圖 4：系統控制迴圈及流程選擇轉換圖

畫時間就增加了 $m \times T_{interval}$；而系統更新畫面的過程則是會逐漸消耗佇列累積的動畫時間。

五、 系統運作流程

我們系統運作的流程，是由計畫階段與執行階段輪流進行。圖 4(a)為系統控制迴圈。控制迴圈中，分別有三種不同的流程走法。一開始會先讀取使用者的輸入，進入到流程選擇。A 的流程是計畫流程，首先將使用者的輸入放入全域的路徑計畫器中，為化身規劃一條沒有碰撞的全域路徑。再來透過局部運動計畫器，為這條全域路徑搭配局部運動，並且把計畫結果輸入到關鍵格佇列中。

進行完 A 的流程後，下一回合會直接進行 B 的流程。B 的流程是執行動畫播放的流程，主要的工作是從佇列中取出資料進行畫格更新。由於是即時的系統，會根據真實世界經過的時間，從關鍵格佇列中去內差出這一次要更新的組態，如果這個組態是合法的(沒有和環境物體發生碰撞)，就進行畫格更新。如果更新過程中，關鍵格佇列空了，就會進行 C 重置的流程，下一輪再讀取使用者的輸入重新計算，重新進入 A 的流程。如果畫格更新沒有失敗，B 的流程會一直重複，直到有超過一個關鍵格被消耗後，下一回合才會再回到 A 的流程。當使用者停止輸入時，則進行 C 的流程，直接重置。圖 4(b)為整個流整選擇的轉換圖。

(一) 局部運動計畫

第三人稱化身的運動產生，是由全域計畫器計算出合法的全域路徑後，再交由局部運動計畫器產生手部的運動。全域路徑計畫器產生避碰路徑的方式，可在智慧型瀏覽介面的研究中找到，細節請參考[13]中的作法。在此我們假設系統能順利產生合法的全域路徑，進一步交由局部運動計畫器，即時產生搭配此路徑的手部運動。

(二) 「組態-時間」空間街圖

對上半身而言，一般運動計畫問題所定義的組態空間，是站在原地不動的情況下定出來的，但是現在我們所需計畫的物體必須在一條路徑上移動，所以搜尋空間必需重新定義。假設路徑是以時間函數 $f(t)$表示，則 C_x就變成了 $C_x \times t$,或寫成 $C\text{-}T_x$。子組態空間從原本的 n 維變為 n+1 維，增加了時間這一個維度，我們稱此空間為「組態-時間」空間(configuration-time space, C-T Space)。而我們在之前的章節提過，在每個 C_x 中皆可置入 R 在 C_x 中的活動能力圖 G_x。當 C_x 延伸成 $C\text{-}T_x$ 時，G_x 也跟著延伸成 $G\text{-}T_x$，我們稱 $G\text{-}T_x$ 為「組態-時間」空間街圖(configuration-time space roadmap, C-T space Roadmap)。

假設化身的腰部只擁有一個自由度，腰部可以向前彎曲，我們可以在 C_{waist} 中建立活動能力圖 G_{waist}。當路徑決定後，我們對腰部組態搜尋的空間就從 C_{waist} 變成 $C\text{-}T_{waist}$。以離散化後的時間來看，$f(t)$, $t = t_0, t_1, t_2 \cdots t_n$，在每個每個時間點 t_i，都有一個在 t_i 時對應的組態空間 C，在 C 中，也會有一份當時的活動能力圖 G，而 $G\text{-}T_x$ 就是所有 G 串連而成的街圖。

(三) 「組態-時間」空間街圖搜尋

雖然我們的搜尋空間是 $C\text{-}T_x$，但我們在搜尋過程中，其實是以 $G\text{-}T_x$ 做為搜尋空間的代表，這和運動計畫中街圖法採用的精神類似。除了先前提過的兩個限制，碰撞避免與盡量跟隨目標函式 $GF_x(t)$之外，在 $G\text{-}T_x$ 中的搜尋必須再加入兩個限制。第一個限制是搜尋出來的路徑，在 t 的組態部分必須是單調遞增的(monotonic increase)，因為時間是無法倒退的。第二個限制是組態節點在兩相鄰 $G\text{-}T_x$ 上移動限制。從 t_i 到 $t_{(i+1)}$ 時間，在 G 中的某個組態節點 q，只能移動到 G 中的鄰居節點集合 CTN 中的一點。CTN 的定義如下：如果 q 屬於 q 的相鄰節點，或是 q 本身， 則 $q \in CTN$。也就是 q 在下一個時間點只能往現在的鄰居節點移動，或是選擇停留在原地。以圖 5 來舉例，圖中的(b, t_1)在下一個時間點，只能選擇移動到(a, t_2)、(b, t_2)或(c, t_2)。第二個限制主要是要限制局部運動移動的速度在一個範圍內。同樣以離散化後的時間來看 $G\text{-}T_x$，若 $f(t)$, $t = t_0, t_1, t_2 \cdots t_n$，加上這兩個新的限制後，$G\text{-}T_x$ 可以視為一個深度最深為 n 的有向無循環圖 (directed acyclic graph, DAG)，而且這個 DAG 只有一個起點 s，就是在 t_0 時，R 在 G 的組態節點。我們稱加入限制後的 $G\text{-}T_x$ 為 $G\text{-}T_x$ DAG。

搜尋空間與限制決定後，接著是搜尋的目標與策略。對於局部運動來說，我們的搜尋目標是從 $G\text{-}T_x$ DAG 中，找到一條深度為 n 的路徑。如果存在，就代表可以為這段全域路徑搭配上一段合法的局部運動。我們從 $G\text{-}T_x$ DAG 的起點 s 開始，以深度優先搜尋(DFS)

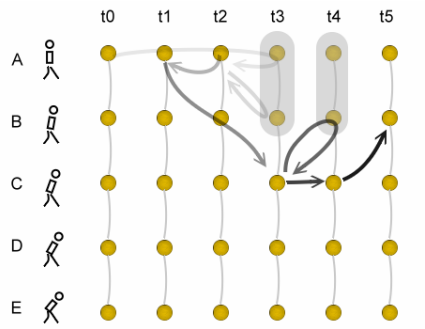


圖 5：「組態-時間」空間街圖搜尋示意圖。
搜尋的順序如同箭頭顏色由淺至深的順序。

的方式，盡量選擇離 $GF_x(t)$提供的 g_x 較近的節點前進，如果能夠走到深度為 n 的地方，代表搜尋成功；否則再以回溯的方式進行調整(backtrack)。當每次需要回溯之前，會先嘗試隨機擴展活動能力圖。在目前失敗的地方，隨機找一些原本不在活動能力圖上，而且距離小於 r 的新組態加入。如果可以加入成功，那代表 $G\text{-}T_x$ DAG 也會跟著擴展，那我們就往剛剛新擴展的組態移動，看看能否透過新的組態繼續朝深度為 n 的地方前進，若還是不行再進行回溯。如果還是搜尋不到長度為 n 的路徑，就代表失敗，無法為這一整段全域路徑搭配上 x 這個自由度群組所屬的局部肢體運動。

圖 6 為根據我們的搜尋目標與策略所設計的搜尋虛擬碼。傳入的參數是 $G\text{-}T_x$ DAG 的起點 s 與搜尋的路徑長度 n。首先要準備一個 stack，用來存放 DFS 搜尋過程中展開的節點。另外要準備一個存放路徑的陣列，path。首先把 s 丟入 stack，開始進行搜尋的迴圈。8～9 行，每次會從 stack 取出節點，根據節點的深度，存到 path 中，接著檢查是否已經到要求的長度了。12～15 行的部分，主要做的事情是根據目標函式來排序拜訪鄰居的優先度，放到 next 中。其中 12 行的 Get_Predict_Time(target.depth + 1)，是指取出關鍵格佇列 Q 中， target 的下一個元素的未來預測時間。17～22 行，則是依序取出 next 中的鄰居節點(越先取出來的表示離 g_x 越遠)。測試這些節點如果沒有碰撞，就放到 stack 中。在 24～30 行中，若當 next 的全部的鄰居節點都無法通過碰撞檢查時，就以隨機的方式搜尋新的節點，若新節點可以通過碰撞檢查，就把新節點放到 stack 中，並且執行 Enhance 的步驟把新節點加入化身活動能力圖及重新連結連結化身活動能力圖。

```
Procedure CTRoadmap_Search(s, n) {
1.  path ← MakeEmptyPath ( n );
2.  stack ←MakeEmptyStack ();
3.  target ← null;
4.  desired ← null;
5.  t ← 0;
6.  Push( s , stack)
7.  while (stack is not EMPTY) {
8.    target ← Pop(stack);
9.    path[target.depth] ← target;
10.   if (target.depth = n)
11.     return path;
12.   t ← Get_Predict_Time(target.depth + 1)
13.   desire ← GF(t);
14.   next ← Get_CTN(target);
15.   Sort_Priority_Increase (next, desire);
16.   count ← 0;
17.   while( next is not Empty) {
18.     candidate ← Pop(next);
19.     if (candidate dose not collide at time = t)  {
20.       count ← count+1;
21.       Push(candidate, stack);
22.     }
23.   }
24.   if(count = 0) {
25.     new_node ← Search_New_Neighbor(target);
26.     if(  new_node does not collide at time = t)  {
27.       Push( new_node , stack);
28.       Enhance();
29.     }
30.   }
31.  }
32.  return failure
33. }
```

圖 6：「組態-時間」空間街圖搜尋虛擬碼。

六、 實驗結果與討論

系統在實作上，我們使用了 Java 程式語言進行開發，版本為 JDK5.0。3D 碰撞偵測的部分，我們使用 V-Clip 碰撞偵測演算法的 Java 版本[5][29]。而場景與人物模型的定義，我們選用 VRML 語言。另外 3D 顯示部分我們使用 Xj3D[30]，該元件是以 Java 語言開發的 3D 顯示瀏覽器元件，可以顯示 X3D/VRML 描述的虛擬環境。在實驗部分，執行的機器 CPU 為 P4 2.8G，RAM 為 512MB，顯示卡晶片則是 NVIDIA Fx5200。

(一) 實驗設計

我們設計了四個擺設方式不同的簡單場景(如圖 6.1)來測驗我們的機制。對每個場景，會使用 3~7 這五個不同的關鍵格佇列大小分別進行實驗，藉此檢驗在不同的情況下，局部運動計畫是否能夠發會效用。

為了控制使用者輸入操作上的變因，我們限制只能讓化身前進或後退，而場景中的障礙物，都是擺在化身前進方向的路上，當化身前進時，我們的機制會替化身計畫上半身的動作，讓行進的過程中，化身可以閃避場景中的障礙物。

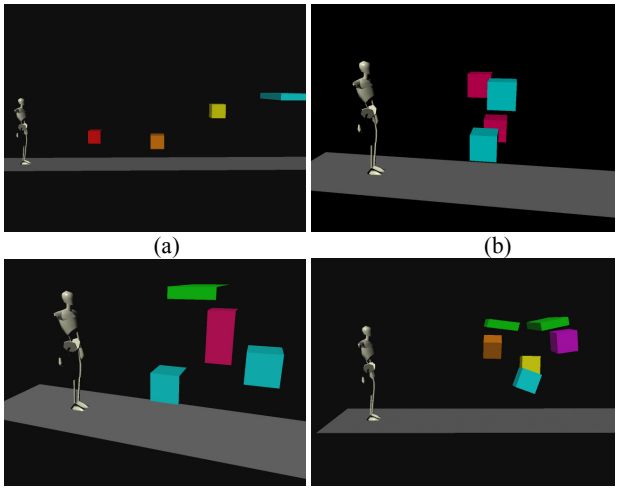


圖 7：效用測試實驗場景

化身模型只要符合 Humanoid 規格的化身模型皆可在系統中使用。四個場景中，我們所設計的難度是遞增的，場景 A 中有四個障礙物，是平均分散在化身會經過的路上，主要是測試單一部位是否能夠避開障礙物。場景 B 中，化身的手與肩膀會同時接觸到障礙物，主要是測試我們所使用的分解式運動計畫方法，是否能夠順利解決不同部位同時發生碰撞的問題。場景 C 與場景 B 類似，但是我們把障礙物在前進方向的排列稍微拉開一些，讓化身通過時，陷入障礙物群的時間加長，看看能否處理不同部位連續會發生碰撞的問題。場景 D 則是四個當中最難的，除了障礙物數目增加到 6 個，我們把障礙物的角度都調整過，讓整個可以通過的空間變的更小，在前進方向的障礙物排列上也調整過，在通過過程中，要閃避障礙物的時間會變更長。

我們對每個場景，都進行五組實驗，五組分別設定不同的關鍵格序列的大小(大小從 3～7)。在一組實驗中，我們會進行三次的「穿越測驗」(分別使用不同亂數種子)，「穿越測驗」是讓使用者控制化身，在 30 秒內，嘗試能不能通過場景中的障礙物群，如果可以通過就算成功。過程中，如果化身與障礙物發生碰撞導致化身被卡住，那使用者必須將化身向前或向後移動，重新嘗試看能不能通過。如果時間超過 30 秒尚未通過，就算失敗。

過程中記錄的數據，第一類是成功與否，包含是否在 30 秒內通過(< 30 sec)，是否一次就通過(straight)，卡住的次數(stuck times)，花費多少時間通過(passed time)。第二類是局部運動計畫呼叫的次數。最後還紀錄了整個過程中碰撞檢查的總次數，及整個過程，是否有維持即時性的效能。

表 1：局部運動計畫器實驗數據

SCENE		A					B				
KeyFrame Queue Size		3	4	5	6	7	3	4	5	6	7
Success	< 30 sec. rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	straight rate	0%	100%	100%	100%	100%	0%	100%	100%	100%	100%
	stuck times	1.3	0.0	0.0	0.0	0.0	12.0	0.0	0.0	0.0	0.0
	passed time (sec.)	8.3	5.0	5.0	5.0	5.0	16.7	2.7	2.3	2.3	2.0
Local Planning	Total	40.0	28.0	27.7	26.7	27.0	78.3	15.0	13.7	13.0	12.3
	Success rate	79.2%	90.5%	90.4%	90.0%	90.1%	60.9%	84.4%	85.4%	92.3%	100.0%
	Failed rate	20.8%	9.5%	9.6%	10.0%	9.9%	39.1%	15.6%	14.6%	7.7%	0.0%
Num of collision checks		8080	6408	6962	7347	8006	13775	3460	3689	3629	3413
RealTime Performance		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SCENE		C					D				
KeyFrame Queue Size		3	4	5	6	7	3	4	5	6	7
Success	< 30 sec. rate	100%	100%	100%	100%	100%	0	67%	33%	67%	100%
	straight rate	0%	67%	100%	100%	67%		0%	0%	0%	0%
	stuck times	2.7	0.3	0.0	0.0	0.3		2.5	9.0	1.0	1.3
	passed time (sec.)	8.0	3.3	3.3	3.3	9.0		7.5	10.0	12.5	17.3
Local Planning	Total	35.0	18.3	19.3	16.7	15.7		37.5	49.0	26.5	32.3
	Success rate	61.9%	69.1%	79.3%	88.0%	95.7%		68.0%	65.3%	84.9%	78.4%
	Failed rate	38.1%	30.9%	20.7%	12.0%	4.3%		32.0%	34.7%	15.1%	21.6%
Num of collision checks		6721	4339	8834	12854	74252		8468	7903	85420	121373
RealTime Performance		Y	Y	Y	Y	N		Y	Y	N	N

(二) 實驗結果

實驗結果如表 1，數據都是三次實驗的平均值，我們只取成功的樣本來平均(30 秒內通過的都算)，主要是因為失敗的樣本累積的時間較長(30 秒)，如果加入可能會影響到真正成功的數據。為了表示上的方便，我們以(場景,關鍵格佇列大小)來描述表格中的某一行。例如(A,4)，表示場景 A，使用關鍵格佇列大小為 4 的情況。圖 8 及圖 9 所示之動畫為由系統成功產生在場景 C 及 D 之化身運動的範例。

我們觀察到(D,3)~(D,7)都是無法一次通過的，而對於場景 A、B 及 C 來說，只要| Q | ≠3，絕大多數都可以通過。在通過的條件放寬到 30 秒內的情況下，對於 A、B 與 C 這三個場景，使用| Q |=3~7 時，全部都能順利的通過，但是對場景 D，當| Q |=3 還是無法通過，而| Q |=4~7 能夠通過測試的狀態明顯比其他場景少。

另外注意表 1 中的即時性這一欄，可以發現(C,7) (D,6) (D,7)三個狀態下，即時性是 N，代表在操作的過程中，畫面更新率的量測有出現每秒會量測到只有一次的情況，我們說在這樣的狀態下，雖然能夠順利通過，但已經不具有即時性的效能。

綜合上面的觀察現象，我們可以發現| Q |如果太小，會使得我們的機制發揮受到一些限制。| Q |越大，

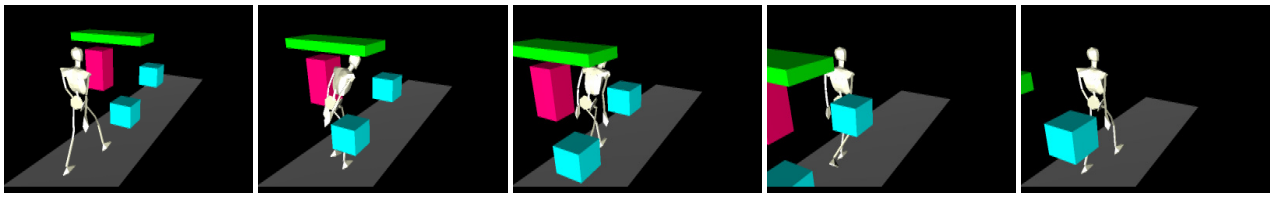


圖 8：化身通過場景 C 的範例

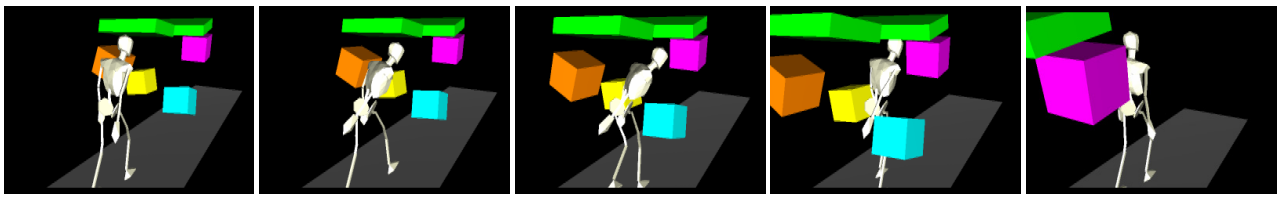


圖 9：化身通過場景 D 的範例

能夠解決的問題數越多，但對於複雜的場景，| Q |太大會喪失即時性的效能。在合理的| Q |大小底下(因機器效能而異，在我們的實驗環境中約在 4～6 之間)，對於 A、B 與 C 這類的場景，即使發生無法一次通過情況，並不會帶來致命的影響，因為只要稍微調整，還是可以很快的通過。對於目前我們使用的分解式運動計畫方法而言，類似場景 C 這種組合的環境，差不多是目前能夠順利處理的極限。

七、 結論

在本論文中，我們將智慧型控制的概念延伸到第三人稱化身上。我們設計了一個三人稱的智慧型化身控制系統，在符合即時運算的效能下，讓使用者在控制化身的過程中，系統自動幫化身搭配能夠保持適當之運動樣式兼具能夠閃避障礙物的肢體軌跡，同時能提升化身的控制上的便利性與動作呈現上的真實性。

八、 致謝

此研究在國科會 NSC 94-2213-E-004-006 計畫的支助下完成，特此致謝。

九、 參考文獻

[1] A.J. Hanson and E. Wernert, “Constrained 3D Navigation with 2D Controllers,” in *Proc. of Visualization '97*, pp.175-182, 1997.

[2] A. Witkin and Z. Popovic, “Motion Warping,” in *Proc. of the 22nd Annual Conf. on Computer Graphics and Interactive Techniques*, pp.105-108, 1995.

[3] B. Salomon, M. Garber, M.C. Lin, D. Manocha, “Interactive Navigation in Complex Environments Using Path Planning,” in *Proc. of the Symp. on Interactive 3D Graphics*, pp.41-50, 2003.

[4] C. Rose, M.F. Cohen and B. Bodenheimer, “Verbs and Adverbs: Multidimensional Motion Interpolation,” *IEEE Computer Graphics and Applications*, Vol.18, No.5, pp.32-40, 1998.

[5] D. Xiao, R. Hubbold, “Navigation Guided by Artificial Force Fields,” in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pp.179-186, 1998.

[6] E.A. Wernert and A.J. Hanson, “A Framework for Assisted Exploration with Collaboration,” in *Proc. of the Conf. on Visualization '99*, pp.241-248, 1999.

[7] J.J. Kuffner and S.M. LaValle, “RRT-Connect: An Efficient Approach to Sin-gle-Query Path Planning,” in *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp.995-1001, 2000.

[8] J. Pettre, JP. Laumond and T. Simeon, “A 2-Stages Locomotion Planner for Digital Actors,” in *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pp.258-264, 2003.

[9] M. Chen , S.J. Mountford and A. Sellen, “A Study in Interactive 3D Rotation Using 2D Control Devices,” *ACM SIGGRAPH Computer Graphics*, Vol.22, No.4, pp.121-129, 1988

[10] Neilson and Olsen, “Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices,” in *Proc. of the Workshop on Interactive 3D Graphics*, pp.175-182, 1986.

[11] P.K. Egbert, and S.H. Winkler, “Collision-Free Object Movement Using Vector-Fields,” in *IEEE Computer Graphics and Applications*, Vol.16, No.4, pp.18-24, 1996.

[12] S.M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” Technical Report, Computer Science Dept., Iowa State University, 1998.

[13] T.Y. Li, and H.K. Ting, “An Intelligent User Interface with Motion Planning for 3D Navigation,” in Proc. of the IEEE Virtual Reality Conference, p.177-184, 2000.

[14] T.Y. Li and H.C. Chou, "Improving Navigation Efficiency with Artificial Force Field," in *Proc. of IPPR Conf. on Computer Vision, Graphics, and Image Processing*, 2001.

[15] T.Y. Li, P.F. Chen and P.Z. Huang, "Motion Planning for Humanoid Walking in a Layered Environment." in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp.3421-3427, 2003.