運動規劃可以被分為兩個關鍵部分，全域規劃與局部規劃。我的研究主要聚焦在局部規劃，但是我還是需要介紹一下兩者再做什麼?

首先，全域規劃就好比手機上的google maps導航，它產生一條清楚的路徑告訴我們如何從現在位置抵達目的地。沿途它會帶領我們避開已知的靜態障礙物，像是國道某路段壅塞，它會建議我們走替代道路；或是某某道路施工，它會帶我們繞開。而局部規劃比較像是車上的駕駛，他必須臨機應變，不斷的做出決策。像是有人突然切到我們前面，駕駛就必須踩煞車；或是突然有野狗燦道路中央，我們必須想辦法避開等等。

由此可見，增強機器人的局部規劃演算法，可以拓展其應對突發狀況的能力，讓他們有機會在高度複雜和動態的環境工作，提升其自主能力。

運動規劃領域已經有幾種主流的局部規劃方式，當中有的比較適合應對不斷移動的障礙，我會依序介紹一下我參考的方法。

首先，DWA和APF都算是發展比較完善的局部規劃方法，但是他們應對動態物體時會面臨比較大的挑戰。DWA方法沒辦法直接考慮移動中的障礙，他需要將該障礙未來可能會出現的位置通通歸納為靜態障礙，然後再啟發式搜尋下一步該怎麼做。而APF會將與障礙物的距離換算為推力，有相衍伸方法會將障礙物的速度納入推力的項次，但是這種方法往往需要依照情境校準，沒辦法靈活的對應不同速度下的障礙物。

據我所知，較有能力面對未知意圖或非合作關係的動態障礙之方法有: 速度障礙法與MPC。速度障礙法又稱VO，他可以直接將障礙物的位置及速度轉換為速度空間中的一道邊界，這做法很有效，但是它假設了物體的速度為常數。所以，當VO法面對在變換速度中的物體時，產生的速度命令變動會很大，有時還會導致失效。而MPC則因為具有預測未來的功能，可以應對複雜行動的物體。但是由於它不像VO法一樣直接考慮物體速度，而是像DWA一樣對物體的未來位置做採樣，因此，這方法往往需要搭配一個長期的預測模型來做一個較為長遠的考慮才能有效避障。而這就很吃重預測的精準性，且計算量往往很可觀。所以，我希望可以結合速度障礙法與MPC的概念，彌補這兩項方法的劣勢。

首先讓我們來介紹一下傳統的VO方法，基本上這個方法就是不斷地決定機器人下一步的速度向量然後移動，接著再決定下一步，如此不斷循環。

怎麼決定這個速度向量呢? 這個演算法的核心，是想要讓機器人在閃避障礙物的同時，仍然朝目標以最快的方式前進，而這就成為一個最佳化問題。

所以，最佳化的設計變數就會是機器人下一步的速度向量v\_next。而我們有一個已知的理想速度v\_ideal，這個理想速度是機器人在沒有其他限制下能朝目標最快速前進的速度向量。最佳化的目標函式，就是去最小化機器人的下一步與理想速度的差異。

而限制式的部分可以看到右手邊的這張圖，這張圖代表著機器人的速度空間。圓圈範圍內是考慮了機器人的最大速度限制後，實際的可行區域。而紅色範圍則是危險區域，象徵著一旦機器人的速度向量落入該區，便有可能發生碰撞。

所以，我定義一個VO函數，一旦速度向量落於危險區域內，函數值便大於0；反之，則小於等於0。於是這個最佳化的限制式，就可被寫為: v\_next必須小於最大速度，且VO函數的值必須小於等於0。至於這個危險區域該怎麼定義，我下一張投影片會介紹。

假設機器人為節點A，當它遭遇了一個物體B，我們可以定義一個A的危險動作動作集合。

怎麼做呢? 直覺上，在configuration space 也就是 C-space 上幫障礙物長出膨脹層後，我們可以從機器人的中心對著膨脹層做兩切線。這個錐形區域便代表著節點A朝著這範圍內的方向移動，都有可能造成與B相撞。

接著，我們轉換一下視角，把這個錐形區域歸納進機器人的速度空間中。現在，這個範圍象徵著所有可能造成AB碰撞的相對速度v\_AB集合。而這個區域又可被稱為碰撞錐，collision cone。

當B的速度為0，v\_AB這相對速度本身就是機器人的速度。但是，當障礙物本身有速度，而我們又想值觀地知道節點A究竟有那些速度向量是危險的話，便需要將這個碰撞錐平移。

平移的方式，是將這個區域的元素，通通加上v\_B向量。而得到的這個新範圍便稱為Velocity Obstacle，也就是VO，代表著所有會造成碰撞的v\_A速度。我們應該避免選擇這個範圍內的速度向量作為機器人下一步的動作。

我簡短介紹一下model predictive control。最重要的一點是，MPC會運用數學模型去預測自身系統未來一段時間內的狀態。比方說我開車在路上，給定一序列的油門、煞車、轉向等控制輸入，10秒內每一個時間點的車輛位置會出現在哪?

我們首先會將一連串隨時間排序的控制輸入當設計變數，並且定義一個符合自身目的的目標函數，比方說最小化能源消耗，或最貼近目標等等。並且，我們可以施加限制式以符合系統動態，像是速度限制或是預防碰撞等等。

接著，一旦最佳化得到結果，MPC便將這最佳控制序列的第一組，也就是下一刻的動作，輸入給系統。

然後，我們便量測系統的真實狀態，以更新數學預測模型，並且再回到步驟一繼續執行下一步的最佳化。如此不斷循環這個流程。

我的方法有點類似MPC架構。首先，演算法會針對周遭的障礙物進行觀測，預測出他們未來的位置、速度，以及告訴我這些預測數據的信賴區間。這些資訊都會被丟到最佳化步驟裡，成為建造一連串VO限制式的參數。另一方面，最佳化的目標函數會讓機器人盡量朝目標點邁進，並且平滑化機器人的速度。求解完成後，演算法會將這一連串速度的第一個命令輸出給機器人讓它移動，然後再重新執行這個迴圈。流程會一直持續循環直到機器人到達終點。下面的投影片我會再解釋流程。

在預測的環節裡，我使用的方法是卡爾曼濾波器。卡爾曼濾波器通常被用來評估一個系統的當下狀態，但是也可以使用它來做短期的預測。它的概念就是我有一個假設的系統數學模型，但是它可能沒辦法精確地描述真實現況；而同時我也會有觀測數據，但是感測器也會有誤差。所以，卡爾曼濾波器會在觀測數據和內部模型中做取捨，看我要相信何者比較多，然後給出當下較有可能的狀態以及它評估的誤差範圍。我在這裡使用的內部模型是常加速度模型，顧名思義，就是我假設動態障礙物短期內的加速度不會改變，然後用卡爾曼濾波器評估障礙物未來的位置及速度，以及該預測的誤差範圍。

以下是這個預測方式的範例，等一下出現在物體前方的紅色圓圈為卡爾曼濾波器預測物體未來會出現的位置，圓圈大小則代表99%左右的信心區間。可以看到該藍色物體實際上沒有按照一定的加速度行走，但是由於我的預測時域不長，且卡爾曼濾波器會不斷靠觀測數據修正，所以預測的結果都還算合理。

我的方法並不像傳統的VO方法一樣只考慮當下，相反的，我需要考慮沿著上升時間形成一組序列的速度向量。也就是說，我最佳化的設計變數會是v1, v2,…一直延續到vn。

而目標函數則由兩個成本函數組成:

1. 第一個是目標點距離的成本函數，它會取機器人接下來的每個位置與目標點的距離之平方作為成本。最小化這項成本可以促進機器人以最有效率的方式移動到終點。
2. 第二項成本函數則考慮了每段速度向量之間的落差，它會取每一步的速度項量與前一步的變化之平方作為成本。最小化這個成本函數可以改善一連串速度命令的平滑性與連續性。

所以，藉由結合上述兩項成本函數，這個最佳化的目標在於從移動效率與速度連續性之間取得平衡，讓機器人能夠順暢的到達其終點。

正如前面所講的一樣，我會使用VO限制式作為讓機器人避免碰撞的手段。