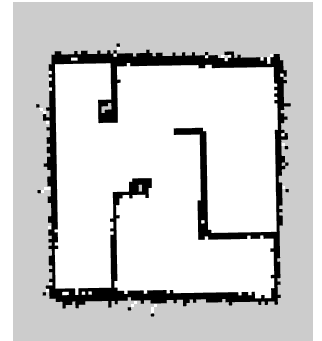
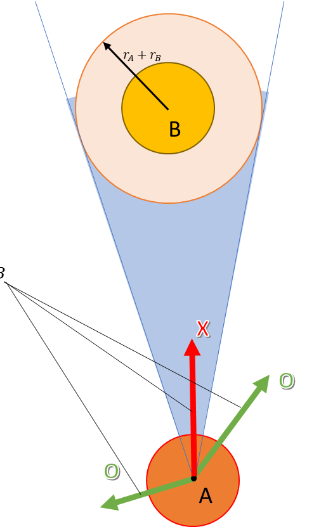
* 首先，我想要解釋一下為甚麼機器人要進行動態避障? 機器人在導航時，通常都會有一個格點地圖已經記錄了已知的所有障礙物。如這張圖
* 
* 但而當機器人實際行走時，可能會有非預期的動態障礙出現在這個地圖中。這個動態障礙可以是出現在路徑上的角錐、四處走動的人或其他機器人。動態障礙是干擾機器人到達目的地的因素之一，輕則阻擋住預先規畫好的路徑，重則造成碰撞。因此，具有動態避障能力對機器人來說至關重要。而這項技術也同時被廣泛引用在自駕車、無人機等領域。
* 在開頭，我想要引用一篇無人機的論文。它將現今能夠進行動態避障的演算法大致分為6個類別。其中一個類別，被稱為幾何引導法，這類的演算法有計算成本相對低的優勢，比較適合用在需要隨機反映、實時反應的動態環境。
* 這個幾何導引法，下面有兩個種類的演算法。一個是碰撞錐理論，而另一個就是速度障礙法，兩個演算法算是同時出現在機器人領域中。以下將對Velocity Obstacle，也就是VO演算法多做介紹。
* 這邊說文解字一下速度障礙法的意思。”速度障礙”，顧名思義就是會在機器人的速度空間中，劃出一個禁止進入的範圍，就像一個虛擬的障礙物出現在速度空間中。如這張圖

所以，VO的概念，就是在機器人的速度空間中，判斷出會導致碰撞的速度集合。機器人應選擇這個集合以外的最佳速度，來避開障礙物，並且到達目的地。

* 那麼，應該如何制定這個在速度空間中的障礙物呢? 我想用這張圖來示意一下它的作法。
* 
* 比方說我們有機器人A與障礙物B。在演算法中，B長出rA+rB的膨脹層是機器人A的中心不能碰到的區域。如果今天B是靜止的障礙物，機器人應該可以輕易的判斷說，如果自己的速度向量落在這個錐形區域的話，應該會發生碰撞。但是，今天B是具有速度的動態障礙物的話，這張圖就會變成說是，如果A相對於B的速度落在這個區域的話，會造成碰撞。
* 但是，我們想要知道的是，A究竟有那些速度會造成碰撞，不是這一坨V\_AB而是我想要知道一坨V\_A。所以，演算法會把這個錐形的區域再加回V\_B向量，就可以得知會造成碰撞的V\_A集合了!這個區域位於速度空間中稱為VO。
* 畫出VO後，演算法就到了選擇速度向量的環節了。原本可能全域的路徑規劃有給出一個速度，但是今天那個速度落在VO中，機器人便需要選擇這個速度以外的、最接近原本規劃速度的新速度向量執行。
* 當環境中有多個動態障礙物時，因為一個B就對應一個VO嘛，速度空間中可能會到處都是VO。對於機器人來說，原本可執行的速度範圍可能為這個區域，現在他被一堆錐形VO通通蓋住了，找不到解了。所以，適當的修剪VO範圍，讓演算法不要那麼苛刻，也蠻重要的。怎麼做? 其實經過推導後會發現，越靠近VO頂點的速度向量，造成碰撞的時機也越晚。所以，我們設定一個時間範圍tou，只要超過這個時間範圍才會造成碰撞的速度，我們不會考慮它是危險速度。那麼，VO就可以被修剪為這個形狀，如此一來，可以找出解的範圍就增加了。
* 然而，原始的VO演算法有一些顯而易見的問題。第一，動態障礙物的速度被假設為在遞迴時間內維持等速移動，因此捕捉非線性軌跡的障礙物時會失真。第二，演算法沒有考慮機器人的運動方程式，如右邊那張圖。理想上的節點可以瞬間改變速度向量，然而現實的機器人的運動並不符合VO演算法的假設，也就無法保證不會碰撞。第三，節點與障礙物都被假設為圓形，沒有考慮機器人在不同朝向下的應對。第四，硬體上，機器人除了需要知道障礙物的位置，還需要知道速度，因此對感測器的誤差容忍較小。
* 接下來，我想要介紹一種VO演算法的分支，稱為GVO演算法，它的主要目的是解決上面問題的第二項，也就是想要將機器人的運動方程式納入考慮來避障。
* 怎麼做呢? GVO並不會在速度空間生成障礙物，而是在控制空間中,產生一個障礙物集合。也就是說，原本VO法是判斷說，有哪些速度會造成我碰撞，我要避開那些速度。現在，GVO演算法則是要判斷，有哪些控制輸入會造成碰撞，我要避開那些控制輸入。這裡舉一個例子，像差速式機器人的控制輸入，可以被寫為左輪速與右輪速，V\_L和V\_R，GVO會告訴機器人甚麼V\_L、V\_R不能用，甚麼可以。
* 這個演算法判斷碰撞的方式，不像前面是用做圖法。GVO是去計算給定控制輸入後，機器人中心與障礙物中心的最小距離，會不會低於兩者的半徑合，藉此來探知是否碰撞，正如這張流程圖所示。我們想要知道給機器人A一個為U的控制輸入後，最小距離是否會小於閥值。所以得先計算A與B的位置函數。再取兩者位置的最小距離判斷該控制輸入是否為危險輸入。
* 第一，B of t是障礙物的位置函數，他與時間的關係可以經由現在位置加上機器人觀測到的速度與時間積分得到。在論文中，B都被假設為依照機器人觀測的速度直線行駛。
* 再者，A of t,u是給定機器人控制輸入後， A隨時間的位置函數。同樣，他也是現在位置加上速度隨時間的積分，然而，關鍵在於，這裡的V\_A以及A of t,u是充分考慮機器人的運動方程式才得出來的速度函數。如同下面差速式機器人，給定一個控制輸入後，位置會隨時間非線性的變化。
* 從以上可以得知，給定A一個控制輸入後，A、B間的相對距離是會隨時間變化的。兩者在有限時間內相對距離可以寫成位置相減再取絕對值。得到這條相對距離函數，我們可以知道給定任一個控制input U，都可以得到一個AB的相對距離與時間的曲線圖，如這張圖。而這條曲線的最小值，就是A、B的最小相對距離。所以，每個U都有一個對應的D\_MIN。
* 這裡論文使用兩種方法取最小值。在面對簡單的相對距離函數，我們可以直接將它對時間做偏微分。然而，面對複雜的d of t,u，論文的做法是以一個time horizon內有限個時點作為sample，然後取最小值，如下圖一樣。
* 當A與B的最小相對距離小於R\_A+R\_B，便可以斷定A、B會碰撞，反之則不會。因此，我們就可以將所有危險的U\_I歸納為控制空間中的GVO集合。
* 這邊舉一個例子，如果A是全向式的節點，那它的控制輸入可以直接寫為V\_X和V\_Y，也就代表說，它的速度空間和控制空間是相同的。所以在這個情境中，可以清楚看到GVO是一個錐形，也就是上色處，正如同VO的範圍。此外，我們還可以知道，越靠近頂點，碰撞的時機就越晚。
* 本篇論文使用車輛模型做實驗。車輛模型的控制輸入有兩項，一項為前輪轉角PHI，另一項為後輪前進速度V\_X。
* 我們可以依照車輛模型的運動方程式將車子的速度向輛表達為X\_DOT、Y\_DOT，以及PHI\_DOT，並將他們積分，取得車輛機器人的位置函數。
* 有了A與B的位置函數後，便可以得知兩者的相對距離公式。然而，由於距離的函數變複雜了，不見得能求到偏微分後等於零的點，這篇論文採有限個U作為控制輸入，並用取樣的方式找出每個距離函數的最小值，並將之視為D\_MIN。如果D\_MIN小於A、B的半徑和，便會被歸納為GVO。
* 下圖就是一個GVO的例子，X、Y軸分別代表了前輪轉向與後輪速度的控制輸入，Z軸則是給定U後，對應的最小相對距離。粉色區域代表著相對距離小於閥值，也就是GVO的區域。
* 原本，車輛A應該有一個最快到達目的地的控制輸入，但是它可能位在GVO中，所以，需要選擇一個與之差距最小的U，做為機器人下一步的控制輸入。
* 下圖是論文示意機器人A採用GVO演算法後，如何閃避動態障礙物B。
* 最後，這篇論文的實驗結果呈現出以下幾點:
* 第一，上面三張圖的藍線是關於GVO演算法的計算耗時。論文中實驗了不同參數，分別依序為障礙物數量、控制輸入的採樣數量和time step。對演算法實時計算能力的影響。結果顯示在一定範圍內(障礙物數小於15、控制輸入採樣數小於40、time step小於0.1~0.3秒),演算法仍能以實時的計算性能運行。這表明GVO演算法具有一定的實時計算能力。
* 第二，圖中的紅線代表著機器人到達目的地的耗時，結果顯示,控制輸入採樣數增加會縮短機器人的移動時間,可見更多的採樣會得到更佳的控制輸入。
* 第三，下三張圖是導航成功率。實驗也檢驗了在不同參數設置下,機器人成功抵達目標的比例。結果顯示當障礙物數量增加、控制輸入採樣數降低、time step增大時,成功率會下降。這表明適當增加採樣密度、縮短time step有利於提高導航成功率。