移動式機器人虛擬場景定位導航

0610768 陳品彤
0610832 張語庭

一、研究目標

1. 熟悉 ROS 系統的基本架構與使用方法
2. 學習使用 Gazebo 和 Rviz 軟體
3. 了解機器人定位、建圖、導航、避障等原理
4. 創建多障礙物隨機移動之模擬環境
5. 觀察機器人在動態環境中定位導航及自主避障之情形

二、研究過程與方法

1. 定位與建圖

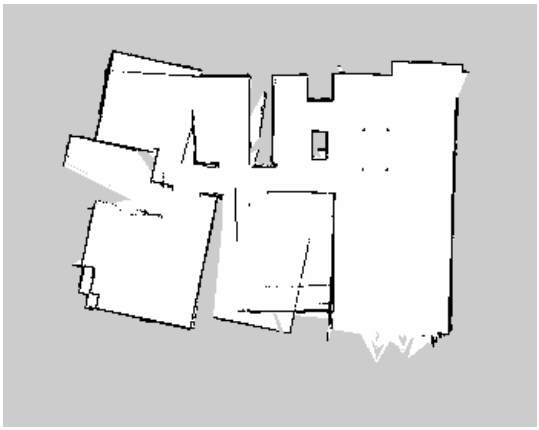
使用 gmapping 做即時定位與地圖構建(Simultaneous Localization and Mapping)，gmapping node 接收 LaserScan 及 Odom 的訊息，建立一個二維的格柵式地圖。

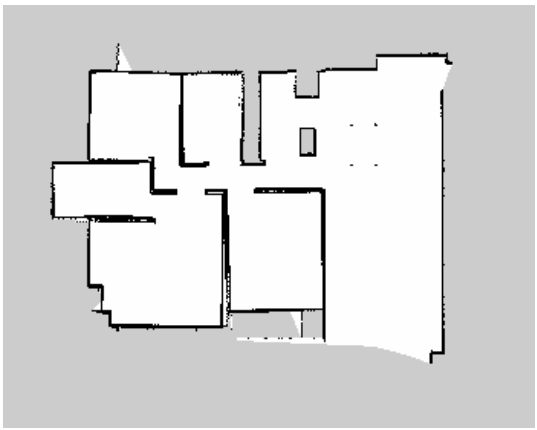
SLAM 演算法用迭代的方式即時推測地圖以及機器人的位姿，而在 gmapping 的 SLAM 演算法中，機器人的位姿是使用粒子濾波器的方式來確定：一開始設定機器人的位置是一團點雲，每個點代表機器人的位置與朝向的可能，最初機器人出現在每個點的機率是相同的。隨著機器人移動，訂閱的里程計數據變化，這些點雲也會跟著一起移動。並且系統會將感測器讀到的資料和每個點看到的地圖做比對，再根據相符合的程度分配每個點的權重，然後得到可能性較高的位姿點，並刪掉可能性較低的點，保持點數總和不變，並在下一個取樣時間生成下一批點雲。如此重複這些步驟之後，點雲會逐漸收斂，便可大致得出機器人的位置和面向。

▲粒子濾波流程

使用 gmapping 建圖的過程中，我們最初使用的感測器是 Kinect，Kinect 一共有三顆鏡頭，回傳的除了有彩色影像外，也有深度資料，其採用的 3D 偵測技術原理，是透過發射紅外線並經過處理，讓雷射光均勻散佈在欲測量之空間中，然後利用紅外線攝影機記錄下隨機反射形成的斑點，再交由晶片計算換算成 3D 深度影像，然而對於建圖而言精度較為不足，距

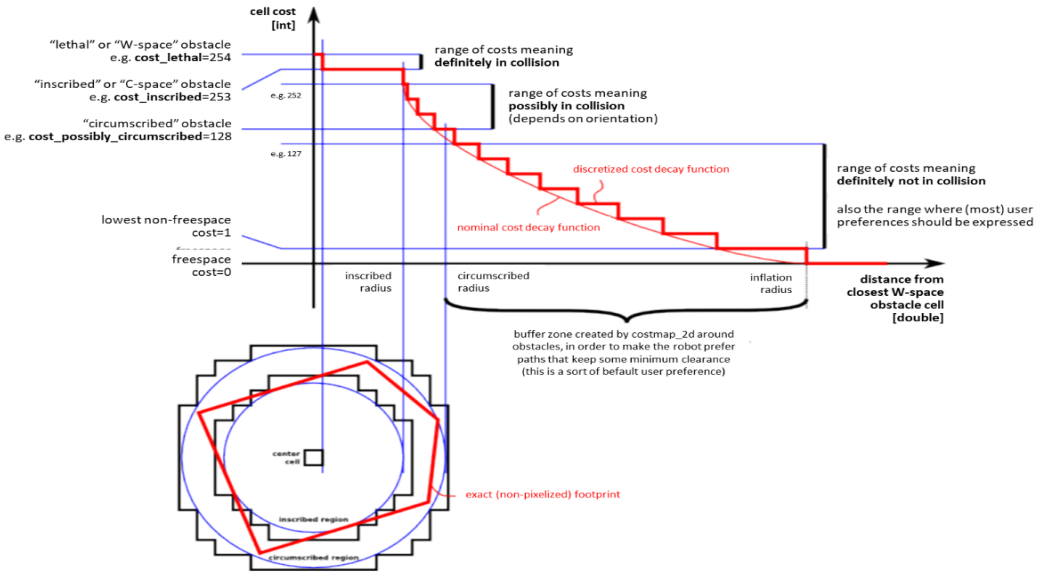
離 3.6m 時，精度約為 3cm，所以構建地圖的後半程誤差逐漸累積導致地圖出現歪斜。後來我們改為使用 RPLidar 進行掃描，RPLidar 的建圖原理則是利用傳感器測量雷射訊號從發出到反彈回來經過的時間，再換算成與障礙物間的距離，距離 6m 時，精度<0.5cm，而且 RPLidar 有快速覆蓋大面積的優點，可以一次獲得大量的訊息，在建圖的過程中有更多資料可以做比較降低誤差，就可以改善地圖偏移的情況了。





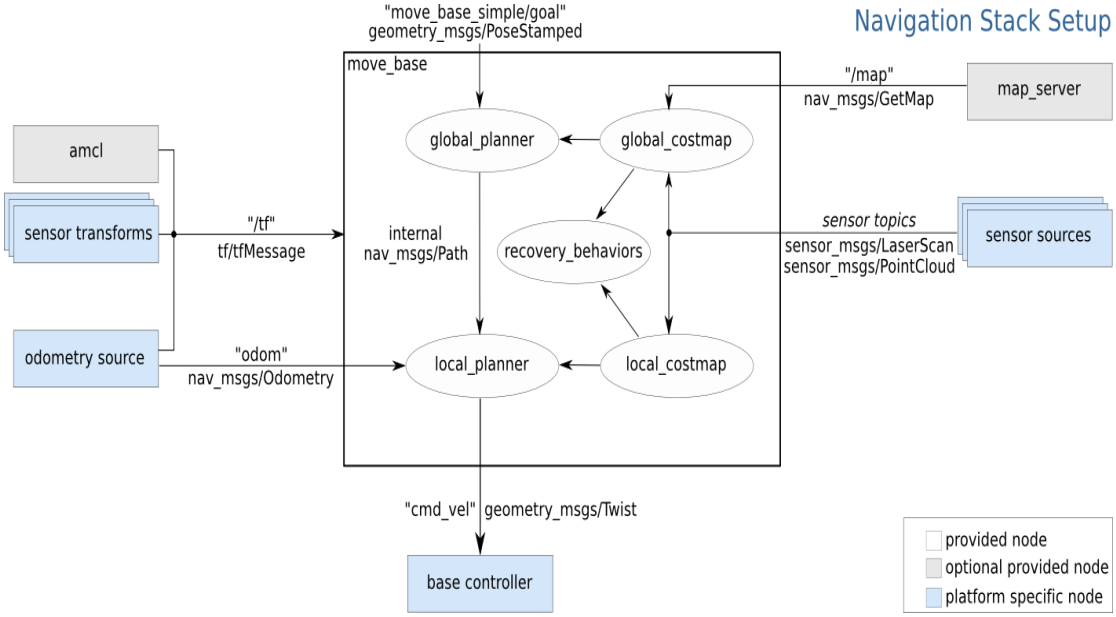
2. 定位導航與避障

基於前面建構好的格柵式地圖，由 map server 交給 move_base 這個節點建立一個全域的 costmap，costmap 同樣是二維的格柵式地圖，每個網格內存儲 0~255 的值，代表障礙物存在在該網格的機率。除了 costmap 之外，為了將機器人的底盤在導航過程中簡化為一個點，還會在 costmap 上面增加一層 inflation layer，將機器人底盤可能造成的碰撞轉嫁到障礙物的 inflation layer，如此便可將機器人視為一個點，去做路徑規劃。機器人使用基於粒子濾波器的 AMCL 演算法進行定位後，會尋找 cost 為極小值的區域，並考慮每個區域與目標點的距離，作最佳的路徑規劃。



除了全局的導航功能外，move_base 也提供實時的區域路徑規劃，根

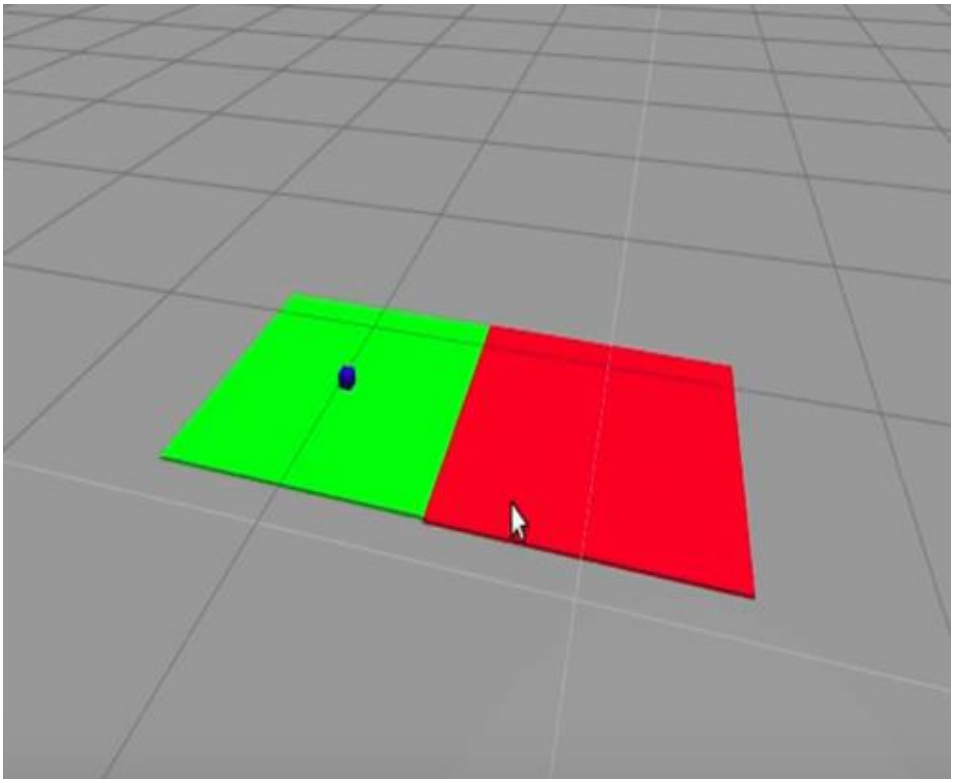
據機器人當前所在區域附近的新障礙物規劃閃避的路線。因此我們規劃在靜態的環境中，加入動態的障礙物，希望能夠觀察機器人在模擬環境中能否藉由 move_base，實現避開動態障礙物的功能。

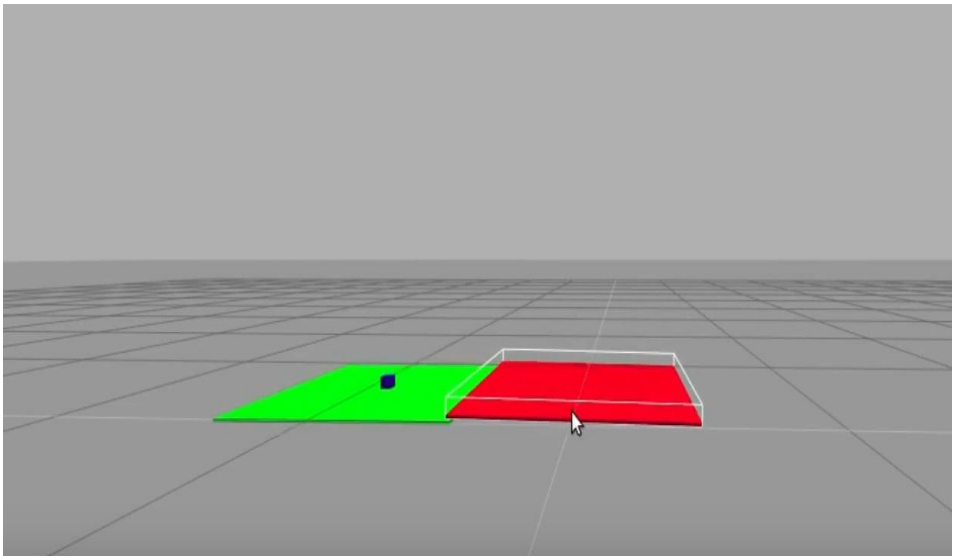


▲navigation stack

3. 在模擬環境中加入隨機移動的障礙物

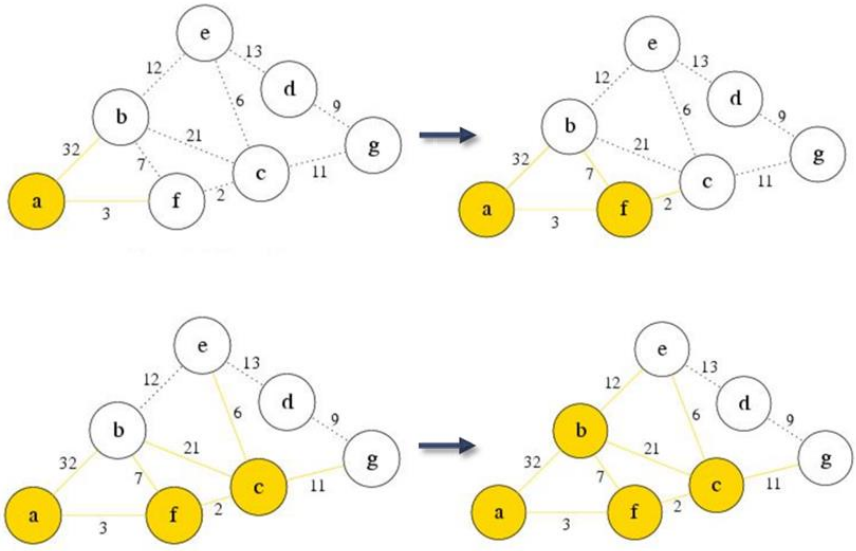
參考 plannar_mover package，將障礙物設計成有兩個關節的機器人，兩個關節分別可在 x、y 方向伸縮，讓障礙物可以到達 z=0 平面上的所有點，而後再撰寫 publisher 控制其移動。





▲圖上的紅色、綠色方塊分別為 x、y link

為了讓障礙物在環境中也能避開障礙隨機移動，我們先在地圖中找好幾個確定沒有障礙物的點(這次總共取的是五個點)，畫好點到點之間的路線，並隨機指定障礙物要前往的目標點，再透過 Dijkstra 演算法規劃最短路徑。



▲Dijkstra's algorithm

Dijkstra 演算法的流程，簡單來說會先以某點為起點，然後尋找該點可以直接到達的所有點，計算並記錄兩兩之間的距離，再選擇最近者為新的起點，重複執行上述動作，記錄下從起點累加的距離，當遇到重複點時選擇更新最短路徑，直到紀錄完整所有節點與原點間的最短路徑。

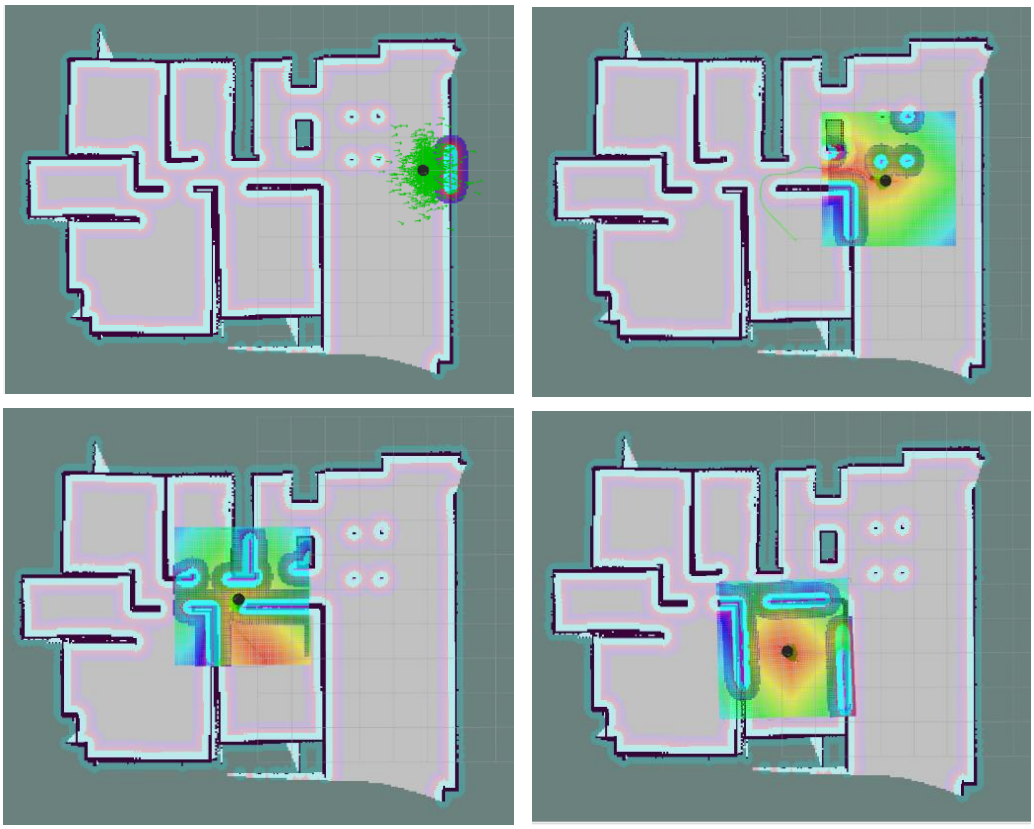
就如同前面所提到的，gmapping 的建圖類型為二維的隔柵式地

圖，故理論上來說，可以透過該演算法，延伸應用至整個格柵式地圖，尋找障礙物與任一網格間的最短路徑。

三、研究結果

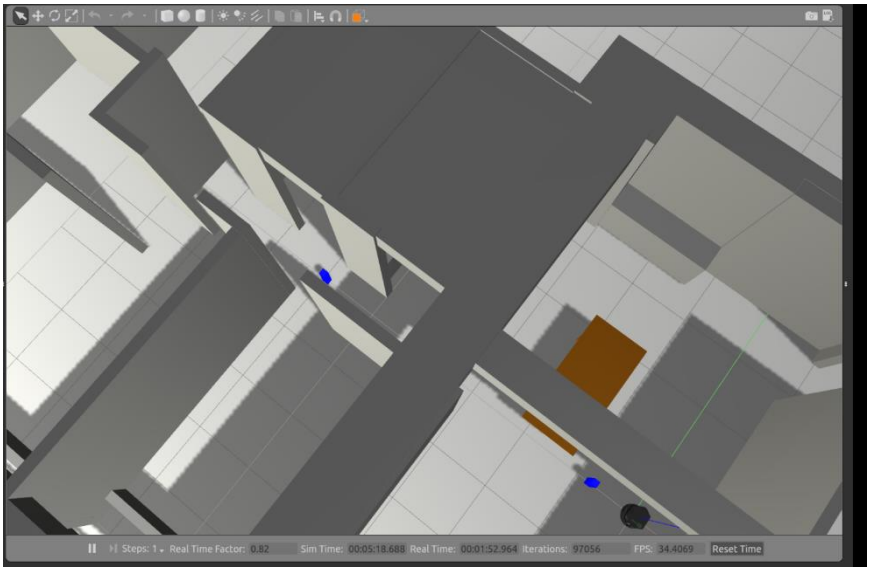
實際操作 Navigation stack 的建圖與避障導航功能，可以觀察到如下圖機器人附近的綠色箭頭會隨著時間收斂，象徵其正在進行粒子濾波的過程，逐漸透過迭代確認機器人當前位姿。

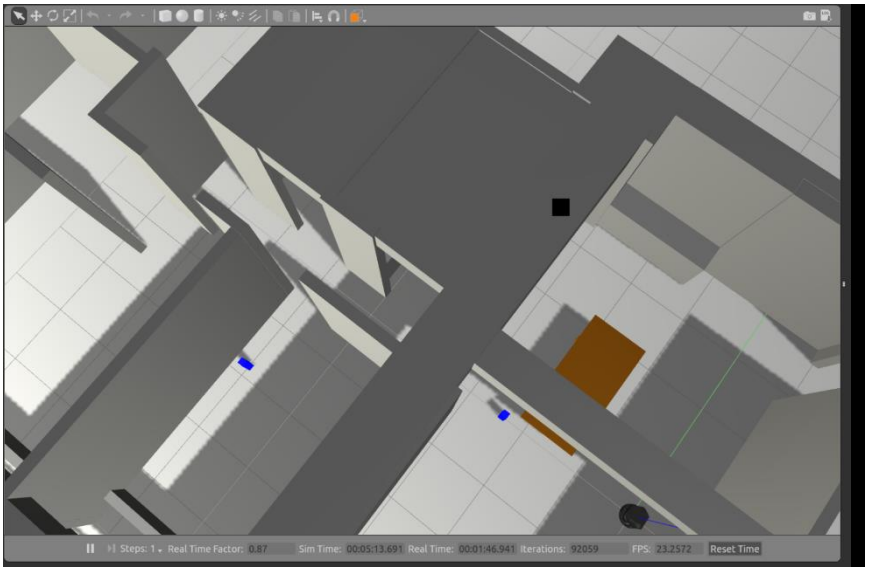
此外，從圖中可以看到由紅到藍的漸層，因為 local_planner 在進行局部路徑規劃時，除了會考慮前文提到的與障礙物相撞的機率 cost 值以外，還會綜合到達目標點之路徑距離做計算，然後選擇總和最小之路徑移動，故圖中的漸層就是反應其經過計算後傾向移動之規劃路徑。



▲turtlebot 自動導航過程

經過修改程式，我們成功創造了一個基本的動態環境，障礙物移動的過程中由電腦隨機指定我們預先選取的幾個目標點，再透過 Dijkstra 演算法程式規劃出最短路徑，然後將路徑中經過的 xy 座標分別發佈給 x 關節和 y 關節，並在運動中訂閱各關節當前位置，不斷檢查是否移動到各個路徑上的點，直到抵達最終目標點則重新發佈。





下圖為檔案運行中各節點間發布及訂閱消息的關係圖，圖中 robot1 跟 robot2 就是我們設計的兩個隨機移動障礙物，可以看到他們發佈 topic 為 command 的目標點消息給 gazebo，然後向 gazebo 訂閱 topic 為 state 的動態位置消息。讓我們撰寫的 publisher 可以確認兩個機器人

在模擬環境中的實際位置，並決定是否要繼續發布消息。

整個檔案執行的流程大約是，由 move_base 擔任導航功能中的主要角色，接收 gazebo、amcl、robot_state_publisher 等節點的消息做路徑規劃的演算，然後發布消息給 nodelet_manager 做配置，統整完成後，再由 nodelet_manager 發布消息給各節點，完成整個導航功能運行的一個循環。

四、參考資料

1. ROS Navigation Tuning Guide
   http://kaiyuzheng.me/documents/navguide.pdf
2. ROS Wiki:documentationhttp://wiki.ros.org/Documentation
3. python 入門教學課程 https://www.youtube.com/playlist?list=PL-g0fdC5RMboYEyt6QS2iLb_1m7QcgfHk
4. 粒子濾波（Particle filter）算法簡介及 MATLAB 實現
   https://blog.csdn.net/qq_27923041/article/details/56008756
5. 鐳射 SLAM 導航系列(一)SLAM 與導航基本原理
   https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/557069/
6. gmapping 原理圖解
   https://www.jianshu.com/p/7a159d950d4b?fbclid=IwAR32eFMBP36h013thZyu7EaODcHHg42chJvnVQ5N7q6YYPgj5LW2Jf91H-w
7. Tutorial: Using a URDF in Gazebo
   http://gazebosim.org/tutorials/?tut=ros_urdf&fbclid=IwAR1y-tucJG86MsQCCfHQK80tEy0m6ehd7JkwqfeuAlmS5zzCRvSsToFuugw#The%3Cgazebo%3EElement
8. My learning note https://medium.com/%E6%88%91%E7%9A%84learning-note
9. plannar_mover
   https://bitbucket.org/theconstructcore/plannar_mover/src/master/plannar_mover/urdf/plannar_mover.urdf?fbclid=IwAR3YbbqbK5ryBZQ7LJv16_o82Gkg1TCsHGJPKWvrB_r4uYebR9eTj-9_Zsc
10. 用 Python 實現 Dijkstra 算法用來尋找兩點之間的最短路徑（Implementation of Dijkstra in Python）
    https://blog.csdn.net/DavyHwang/article/details/46552655
11. 代克思托演算法（Dijkstra's algorithm）
    http://nthucad.cs.nthu.edu.tw/~yyliu/personal/nou/04ds/dijkstra.html