第二天的PPT的有些參考源在這

<https://pojenlai.wordpress.com/2015/07/16/ros-navigation-stack-%E7%B0%A1%E4%BB%8B/>

<http://roscon.ros.org/2014/wp-content/uploads/2014/07/ROSCON2014_DLu.pdf>

在使用navigation時通常會配 amcl (adaptive Monte Carlo localization )

注意: amcl不是包在navigation 裡面的!

Ros裡所有的事件都跟tf有關 tf : 自己的座標(姿態)

投影片第五頁、第六頁

吃了哪些topic，topic餵完後，產生output :

//global costmap 我比較不確定 感覺它可以沒有)

global都是不考慮系統動態模型(車子多寬多大等等本身尺寸、系統資訊)

Global map 產生只有障礙物的靜態地圖 若只有global map 如果今天地圖長的不一樣的時候 車子就會撞到障礙物

Global costmap 要距離障礙物多少才算安全，根據global map的地圖

Global\_planner根據global costmap 餵給他的資訊，我們在給global\_planner餵出一個goal (目的地)(透過rviz給或是送一個message到move\_base\_simple/goal這個topic(但是move\_base\_simple/goal是單純送GO 之後不理你，若想要她每隔幾秒就回報請用Action API )，golbal planner 就會根據 global map 裡靜態障礙物的位置資訊，畫出一條到達目的地的路徑，也就是global\_path

global\_path 沒符合動態軌跡(最上層那層)

global planner 計算最小路徑 (不care 車子動態模型 如table1-2會撞牆)

local costmap 使用的演算法是“DWA(dynamic window approach)”

<http://blog.csdn.net/heyijia0327/article/details/44983551>

<http://myenigma.hatenablog.com/entry/20140624/1403618922>

local costmap (會根據動態模型 避開障礙物且到目的地)

local costmap input 為Sensor ，主要在偵測有無新的障礙物出現，local planner 根據sensor data餵給他的資訊，決定當前要行走的路徑 ，最後在比較global path的路徑進行修正， 最終的cmd\_vel 為trajectory(符合動態方程式(車子的)的output)

trajectory 符合動態方程式(車子的)的output

cmd\_vel、此車有轉換指令(因為車子不能原地旋轉)

投影片的七頁

Action API :

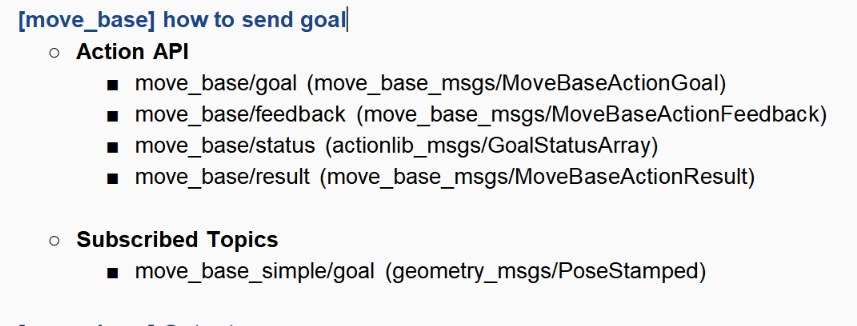
(發訊號後，規定幾秒後 定時回報我，到達目的地後就是沒有回報訊號了)

Example: 假設今天要車子從a點到b點就很適合

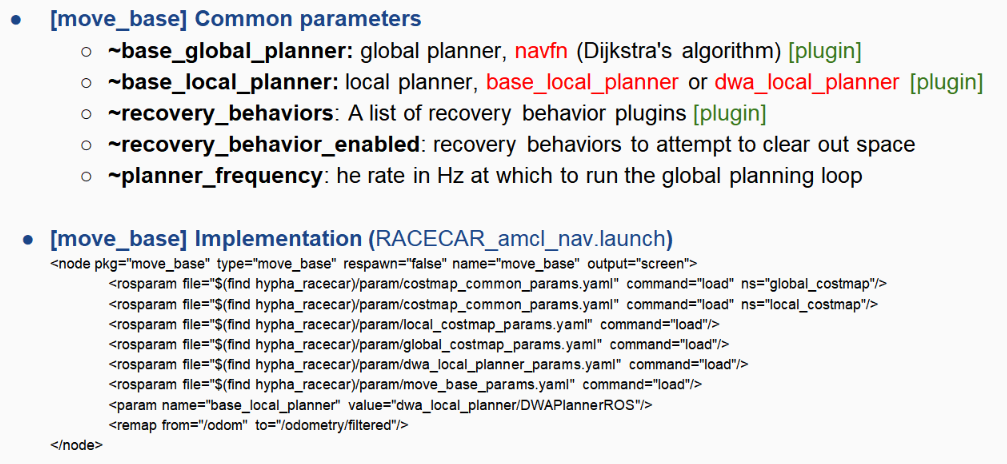
ros action lib 發指令出發 到達目的回報 過程中間會回報給你 ;

action api 就有符合要求，他幾乎等於就是topic 和service 包在一起綜合去寫 ，但action api 有幫你規範完了( 就是不用自己建srv 和 msg topic 之間的溝通)

若是純粹topic，就是固定頻率一直去發送，每分每秒都一直叫車子走



move\_base\_simple/goal :單純送GO 之後不理你



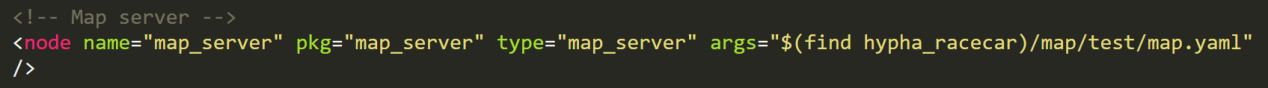
Recovery\_behaviors 我遇到無法前進(會撞到東西)，會先自己（系統）轉一圈之後，附近的障礙物標記會清掉(表示此障礙物是假)，如果能轉，就應該可以前進

在move base 機器人會有範圍，預設假設，機器人周圍沒有障礙物

Planner\_frequency預設FALSE，如果設很高，但會有不知道該往上走還是往下走，雖然可避開大型障礙物 ，此團隊設0.2

//很大的障礙物才會REPLANNER 但是會有很大的問題

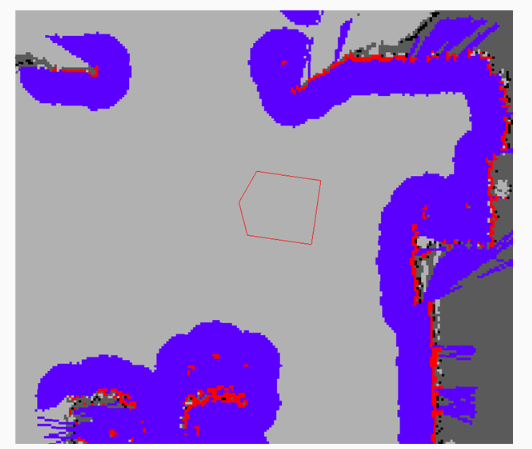
RACECAR\_amcl\_nav.launch



第一次先做global costmap 先”把障礙物標記出來(static map)”，用0-255去實現標記，比如靠近牆壁多近，透過global costmap 去做這些事情，只單純避開障礙物這事情還不夠用，還得根據動態模型去做計算，還得考慮突然出現的障礙物

在位子估測不准就得用，”local costmap(rolling window)” 以車子為中心掃周遭的，在透過global map給他靜態障礙物，glomap costmap 計算離障礙物多遠

Local costmap 把當前障礙物標記出來



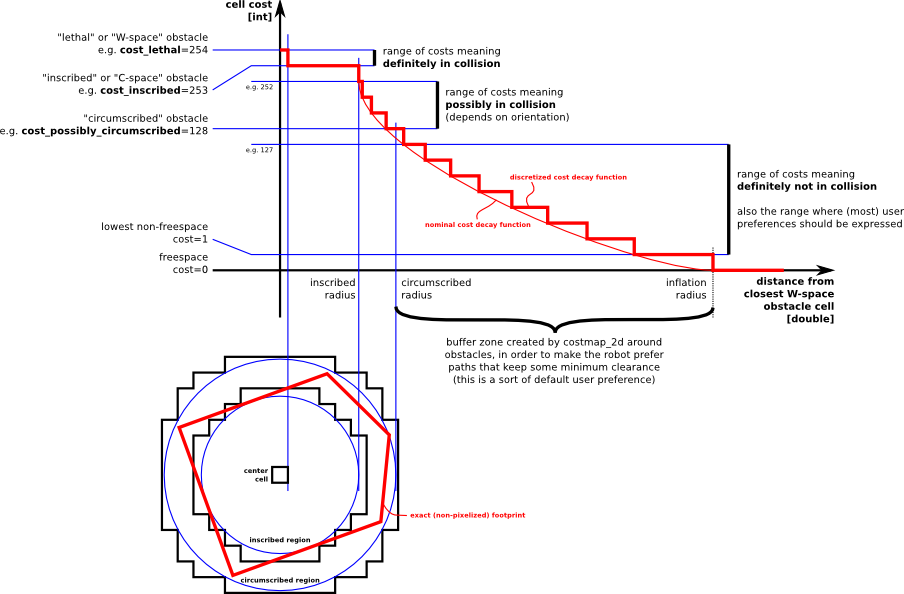
紅點為 global Map建出來(LASER)

藍色為local costmap 顯示，白色是0 ，global planner 所規劃的global path會盡量走0

Local 以機器人看世界

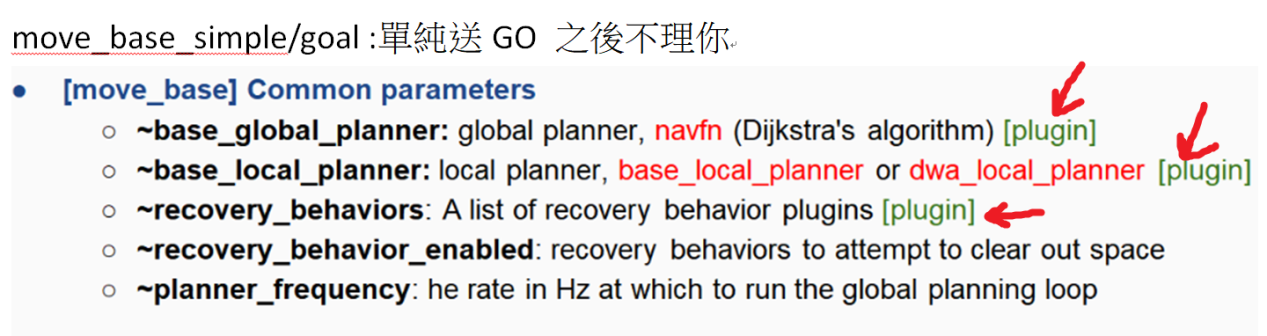
Gloal 全觀

投影片第十一頁、十二、十三頁



機器人到障礙物距離要大於內圓半徑

外徑(第二層)有些時候這個角落會撞到，原本進得去，但是旋轉完就卡住了



Move base 有很多模組 (plug in ) //猜測應該是指上圖紅色箭頭

Rviz 裡map上標註的東西value只有0~255

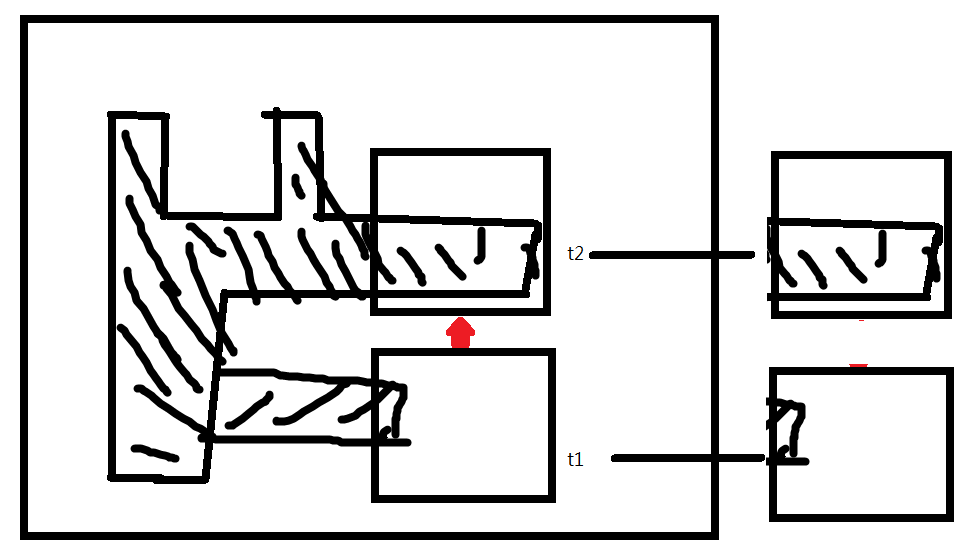
Static map layer(plug in ) 只給gloabla map 用

Obstacle map layer(plug in ) 他吃很多sensor data，若global costmap不吃它，就可以讓gloaba map 無法更新static 障礙物，local costmap 之核心就是這個

inflation layer(plug in ) 用function把255 255 0 列出來

在obstacle 這層做marker or clear預設主持人說忘記了(marker or clear)

Rosrun map\_server map\_saver 儲存當前地圖檔案，此地圖檔案可用小畫家修補(比如玻璃，laser會穿透它，而沒掃到之類的)



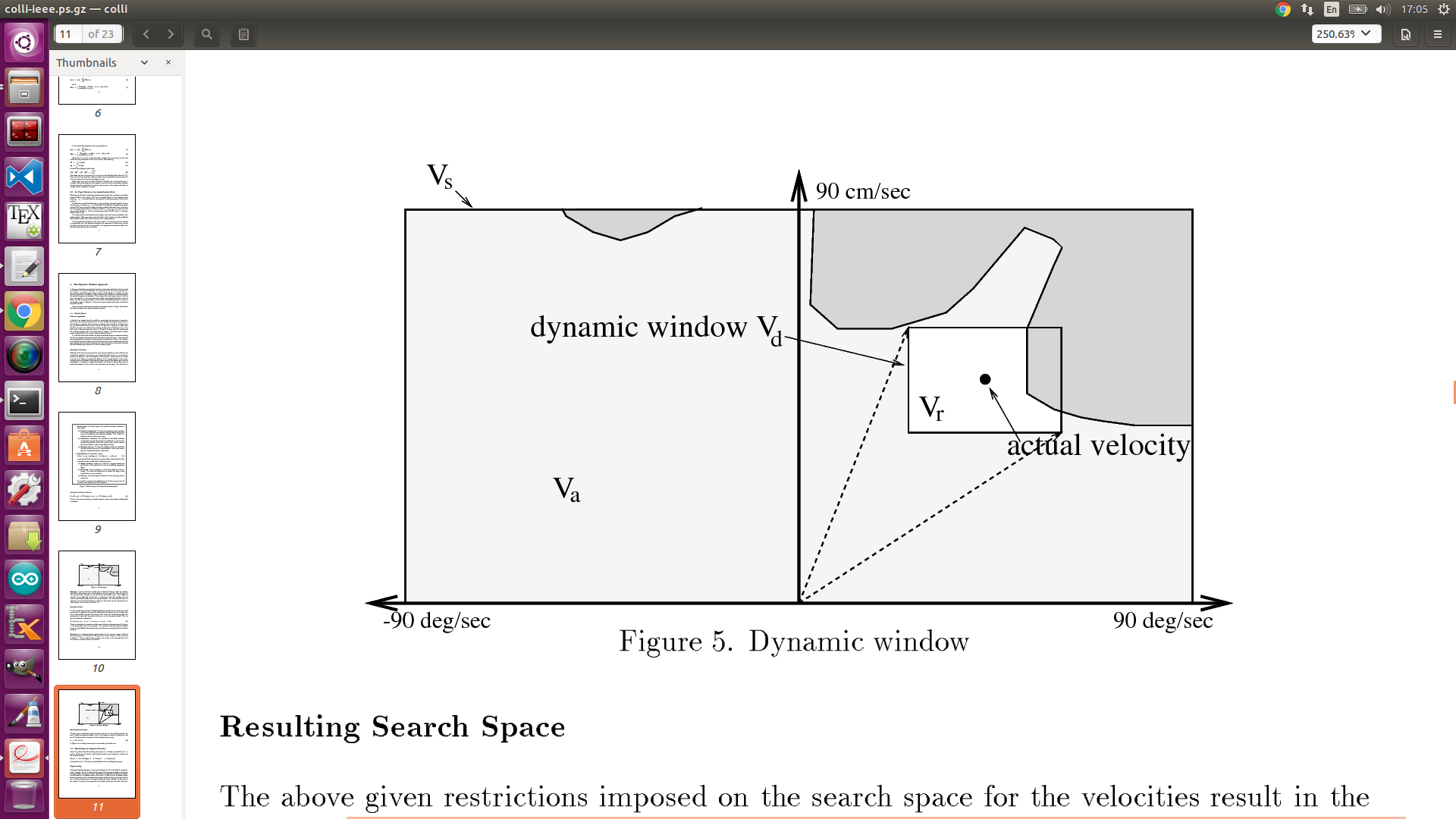
Rolling\_window (map為固定size，有點像CSO右上角的雷達，走到哪就只顯示該區域內可能的障礙物)

Costmap\_common\_params.yaml //global local 共同參數

Gobal\_costmap\_params.yaml

Local\_costmap\_params.yaml

DWA 跟 nagivation 包再一起用 本次是額外寫的control (為了考慮機器動態特性，用DWA(dynamic window approach)

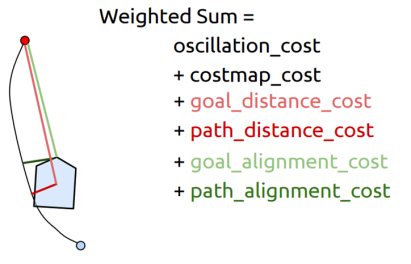
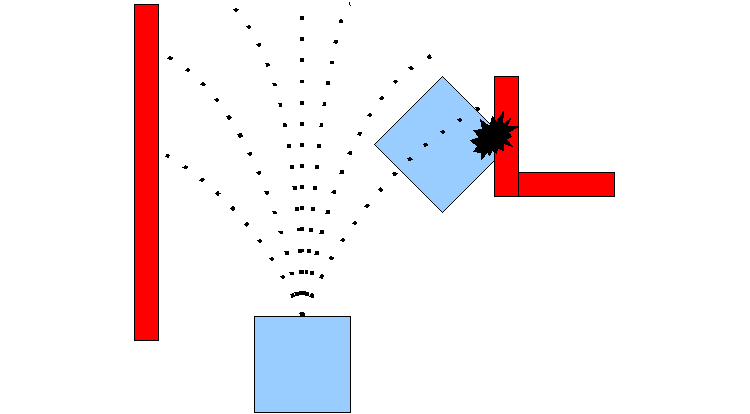


Va沒有包含障礙物的這塊

Vd 原本1m/s，下一秒要100m/s 但是辦不到，此為物理上限制(給範圍) 給window

Vr 在DW(dynamic window)裡的VA

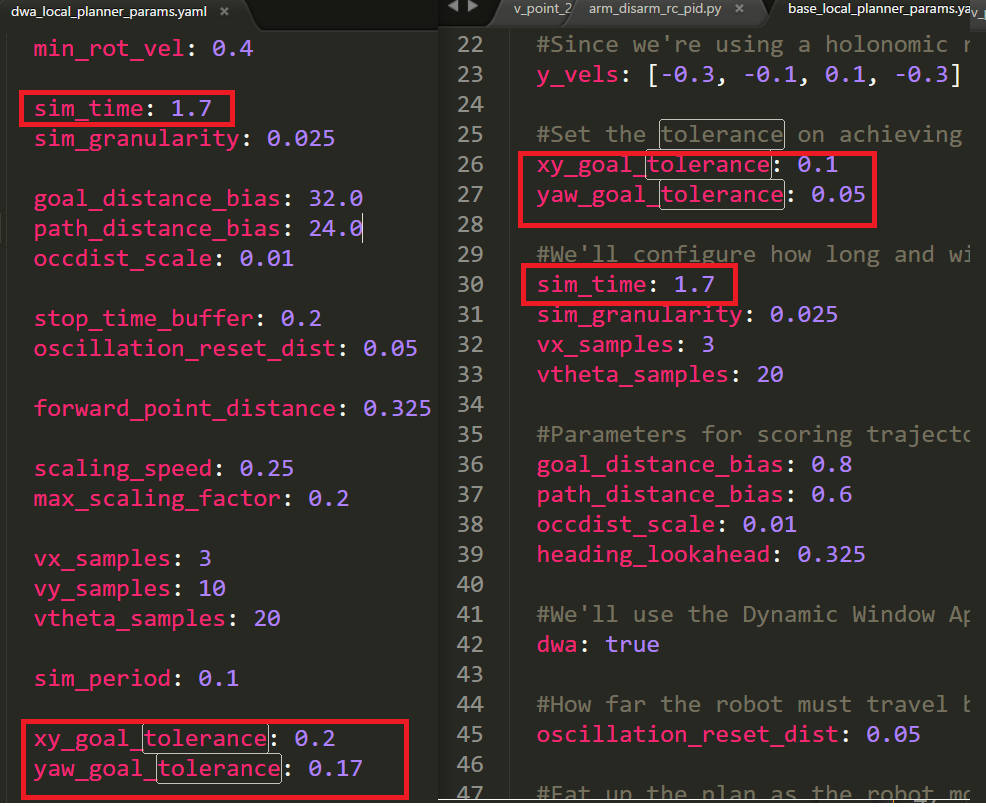
可能有XX條軌跡，再去costmap



<http://wiki.ros.org/dwa_local_planner>

Goal\_distance\_cost希望照著目標走，把此值提高，離目標的路徑會越短

Path\_distance\_cost希望照著路徑走，把此值提高，會越照著原本的路徑走



xy\_goal\_tolerance: 0.2先嘗試到xy附近，可能xy到了用odom 修正，然後又偏差，然後無限輪迴，gola LETCH TRUE，到xy後，就修orientation，不管xy ;預設false，假設odom沒這麼慘(找不到在哪個程式碼20170613)

yaw\_goal\_tolerance: 0.1

LETCH??

在rviz點綠色箭頭(position Essential)，給車子初始值(大概在這個位置)

出事砍掉VM內的navigation\_tutorials去<https://github.com/ros-planning/navigation_tutorials>載回來

**雜項:**

move base 包了 那個綠色長方形

存的地圖(map server) 在靜態記憶體

Plug in(底層控制器) 沒講 要自己查 ， 我猜跟rqt介面上的plug in 有關

Map到odom關係 、odom到 \_base\_link關係

Ros\_ wiki- 他吃了那些

L1 comtrol .cpp 去追

市售rotbot用pwm已算很好

<remap from=”.doon”to = “/更改的鳴子”/>

Other layers:

紅外線 聲納: 這兩個極易受到噪音干擾 (FOV)

Footprint 機器人底層 (四方型)

Robot\_base\_frame 定義機器人map底層

marking: true, clearing: true 有四種選擇

Sim\_time 很重要 模擬時間

20170613