2018 年兩岸暑期學術交流 專題報告

醫院智慧自走車–利用 SLAM 來進行即時定位建圖後導航

作者 張鎧麟

台灣清華大學電機資訊學院 電機工程學系

指導教授 馮輝

復旦大學信息工程與科學學院 電子工程系

2018 年 08 月 24 日

摘要：

近幾年來隨著各個國家發展愈來愈完善，醫療知識的普及與科技的進步使得人類平均壽命提高，也造成了老年化的社會，醫療資源需求不斷地提高，醫院面臨人手逐漸不足的困擾。護理人員不合理的加班、日夜顛倒的排班表所產生的問題逐漸浮出水面，在高壓的環境下，許多護理人員感到心力交瘁，從而選擇離開醫院。為了彌補人力的缺失，我提出一個概念是以智慧自走小車來幫忙搬運藥物、器具……等等，進而減輕護理人員的工作量。智慧自走小車可以利用相對低成本的激光測距雷達（RPlidar），在機器人操作系統（ROS）的基礎上，使用即時定位與地圖建構（SLAM）演算法中的其中一種 Hector Slam 的演算法建構出一個二維的平面地圖，而後利用此平面地圖，設定完初始位置與終點，小車即可自動導航，而途中智慧自走小車還可以透過激光測距雷達即時掃描突然出現的障礙物，重新規劃路線，進而順利地行駛到指定的位置。

第一章 引言：

隨著科技逐漸發展，以前只有在科幻電影裡出現的智慧機器人，現在人們不斷地嘗試將它們帶到現實生活中。為了與現實世界接觸、互動，機器人需要依靠各式各樣的傳感器來接收來自外界的各種資訊，尤其是移動型機器人更是需要外界的訊息來協助它的任務，除了判別自身處的位置，還能精確地知道哪裡是障礙物、哪裡是空曠可走的區域，如此一來才能夠順利自由地移動。移動型機器人需要實現多種功能，然而其中最令人們關注與研究的方面就是自動導航。

根據 J.J Leonard 和 H.F Durrant-Whyte 的研究，自動導航問題可以歸納為三個基本問題：“Where am I?（我在什麼地方？）”、“Where do I want to go?（我想去哪裡？）”以及“How do I get there?（我怎麼到那裡？）”。其中第一個問題就是移動型機器人的定位問題，也是自動導航重要的第一步。而即時定位和地圖建構技術的出現不但解決了定位問題，也相當於針對「先有雞，還是先有蛋？是感測器先感測到機器移動還是機器內部先建構好地圖？」這個問題提出了一項有效的解答。

即時定位和地圖建構（Simultaneous Localization and Mapping, SLAM）最早在 1986 年由 R.C. Smith、M. Self 和 P. Cheeseman 提出，目的在使處於未知環境中的機器人從一個未知的位置出發，通過其攜帶的感測器逐步構建周圍環境地圖，同時在所構建的地圖的基礎上對機器人的位置進行估計和更新。通過即時定位和地圖建構演算法，我們可以更安全地對未知環境進行探索。例如在一些搜查和救援行動中，由於環境的未知可能會使搜救人員面臨一定的風險，因此可以讓擁有即時定位和地圖構建能力的機器人先行探索，在對目標環境有一定的了解之後再讓搜救人員行動，甚至完全代替人類進行工作。其他的像上面提到過的導航功能，或者近年來開始流行的掃地機器人，都可以利用即時定位和地圖構建技術來輔助實現最終的功能。

第二章 研究動機與目的：

近幾年來機器人的發展逐漸從工業用途拓到更多不同層面，例如職場輔助與居家照護等輔助型機器人的應用。輔助型機器人的種類相當多元，在居家照護方面，例如當孩童短暫沒辦法得到父母陪伴時，與孩童互動的學習互動型機器人，例如可以做為攙扶行動不便者的照料型機器人，在發生緊急狀況時及時通知救助人員……等等。在職場輔助方面，機器手臂在工廠的應用大幅提高工作效率，甚至是利用大數據分析病人資料給醫生建議的機器人。在這樣的趨勢下，機器人在未來將逐步地融入人們的生活中。而本次專題提出一個概念是以智慧自走小車來幫忙搬運藥物、器具……等等，進而減輕醫院中護理人員的工作量。

第三章 實驗設計與流程：

一、ROS 簡介與安裝

機器人操作系統（Robotic Operating System, ROS）最早在 2007 年由史丹佛大學的人工智慧研究團隊（Standford Artificial Intelligence Laboratory）所提出，ROS 是專門用於設計機器人與自動化裝置的軟體架構，近年來廣受歡迎。至今國外大部分使用此架構的開發者主要都集中在歐美國家的學術機構與產業界。ROS 的架構主宰著機器的控制、辨識、資料收集等各方面的框架。ROS 之所以廣受歡迎的其中一項原因，就是它能夠有秩序地解決了多執行序（Multi-thread）的艱難任務。並且，一件很複雜的任務，可以在這個架構下拆解成許多子任務，每一個子任務能用一個執行檔負責，最後把所有執行檔，或稱節點（Node）串起來，就完成了一個專案。

為了初步了解 ROS 的系統架構，以下簡單介紹各個基本概念：

〈一〉、節點（Node）：節點是進行計算的進程，也是 ROS 中最基本的組成部分。由於 ROS 是一種模組化設計，而且每個模組都很細微，因此通常來說一個或多個節點只負責一小部分工作，而一個完整的機器人控制系統經常包括許多節點。舉例來說，一個節點控制一個雷射測距器，一個節點控制馬達轉速和輪子，一個節點負責位置定位，多個節點實施路徑規劃，多個節點提供系統的圖像總覽，等等。

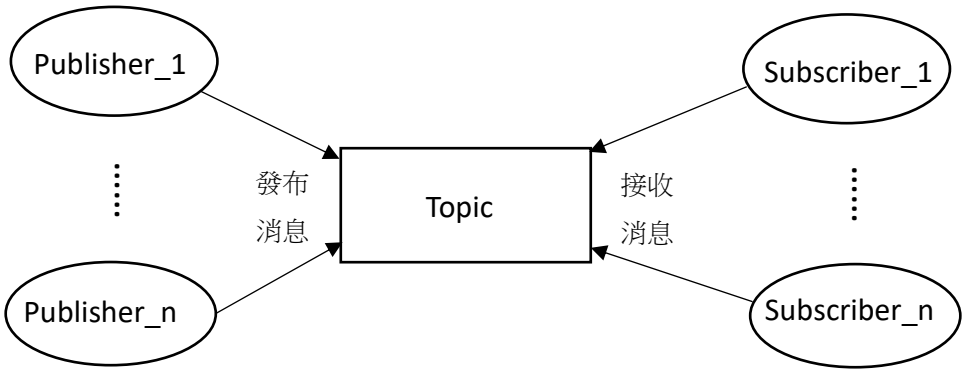
〈二〉、消息（Message）：節點與節點之間通過 messages 來傳遞消息。一個 message 是一個簡單的資料結構，包含一些歸類定義的區，同時支持標準的原始數據類型（整數、浮點數、布林代數，等）和原始數組類型。

〈三〉、主題（Topic）：節點可以發佈消息到主題上（Publish），也可以訂閱主題以接收消息（Subscribe）。所謂的主題就是一個名稱，用來辨認消息的內容。一個節點，如果對某一種特定的資料感興趣，將會訂閱適合的主題。

簡單地說，節點將 messages 發佈到 Topic 上，就像把消息寫在黑板上，而只有訂閱這塊黑板的節點才能接收消息，需要特別注意的是節點之間是不能直接傳遞消息的。下列示意圖中橢圓形為節點，矩形為主題，如圖一所示：

圖一、ROS 基本概念示意圖

而一項複雜的系統通常會由許多發佈者 Publisher 與訂閱者 Subscriber 所組成，如下圖二所示：



圖二、ROS 基本概念示意圖

為了在樹梅派上使用 ROS，本實驗所使用的作業系統為 ROS 官方 wiki 上的映象檔，基於 Ubuntu16.04 的基礎上，預先安裝好了基本的 ROS，參考網址為：https://downloads.ubiquityrobotics.com/pi.html，選擇最新版本下載後放到樹梅派的 SD card 中後開機，安裝好後預設密碼為 ubuntu。

二、RPlidar 簡介與安裝

RPlidar 實體照片可參考下圖三：

圖三、RPlidar A1M1

RPlidar A1M1 是一個低成本的激光掃描測距雷達，有 12 公尺的測距範圍，每秒 8000 次的採樣頻率，接上 USB 接口即可運行。在官方網站 Slamtec 上可找到相關文檔與 sdk、ROS 包的下載位置：
http://www.slamtec.com/cn/Support#rplidar-a1。
使用 RPlidar 時要先啟動 rplidar.launch，終端機指令為：

roslaunch rplidar_ros rplidar.launch

RPidar_ros 是一個節點可將掃描的訊息發佈到/scan 上，而訊息的形式為

sensor_msgs /LaserScan。

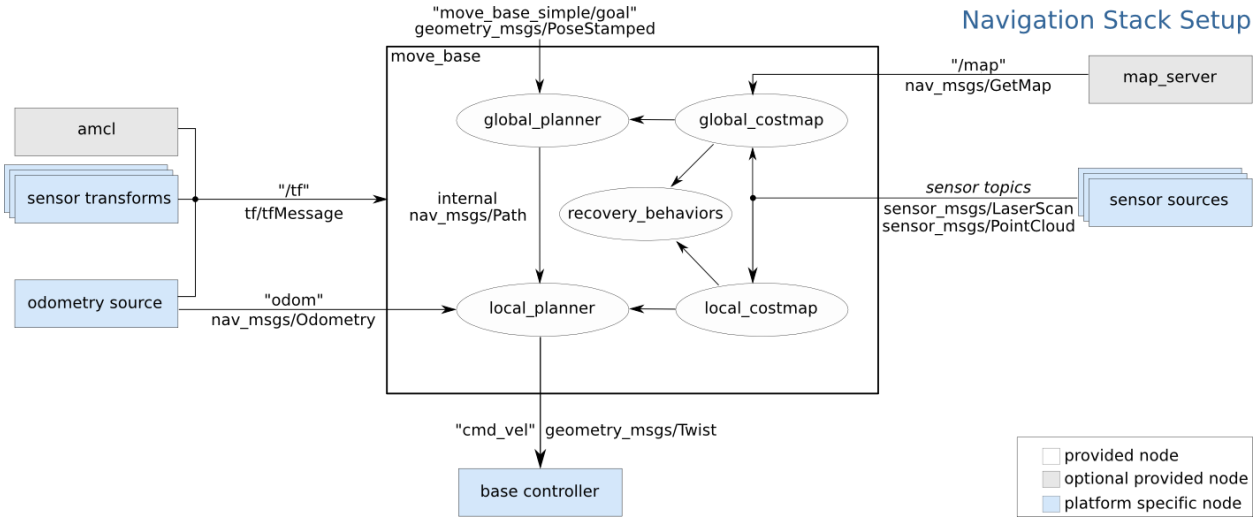
三、Hector SLAM 簡介

Hector SLAM 為 SLAM 的其中一種開源的演算法，相對於其他種演算法（例如：gmapping），Hector SLAM 可以在沒有里程計（Odometry）的狀況下建構二維地圖。激光測距雷達 RPlidar 可將當前這一幀的雷達數據傳遞到/scan 這個 Topic 上，而透過 Hector SLAM 來計算上一幀與當前這一幀的雷達數據，經過演算法來確認距離多遠處有障礙物，經過不斷地計算後，建構出一張縮放後的二維的地圖，實現即時定位與地圖建構的功能。

在 https://github.com/tu-darmstadt-ros-pkg/hector_slam 可下載 package 的壓縮檔，解壓縮後放到 catkin_ws/src 中進行編譯。

四、Navigation Stack 簡介

Navigation Stack 為一個複雜的開源模組，常應用於機器人導航，其中有許多節點和主題，矩形為節點，橢圓形為主題，架構圖如下圖四表示：



圖四、Navigation stack 架構圖

從圖四可以看出最主要的架構為中間的節點 move_base，而 move_base 可以透過"/map","sensor_msgs/LaserScan","/tf","/odom"等主題接收從各個不同節點發布的消息；在 Rviz 上設定完終點後將消息傳到"move_base_simple/goal"這個主題上讓 move_base 接收，move_base 便會依據機器人目前在地圖上所處的位置與終點位置規劃導航路線，最後將控制的指令發佈到"cmd_vel"上。

Navigation stack 的所有 package 可以在 https://github.com/ros-planning/navigation/tree/kinetic-devel 找到。

以下為各個主題（Topic）的介紹：

〈一〉、map：

本主題主要提供 move_base 地圖的資訊，而本次實驗預設醫院裡的格局是固定的，故先利用 Hector SLAM 建構地圖之後使用 Navigation stack 眾多開源

package 中其中的 map_server 將地圖保存下來，而其代碼為：

rosrun map_server map_saver -f mymap

保存的形式為 mymap.pgm 檔和 mymap.yaml 檔。地圖保存下來之後，若要將保存的地圖發佈消息到 map 主題上則需運行：

rosrun map_server map_server mymap.yaml

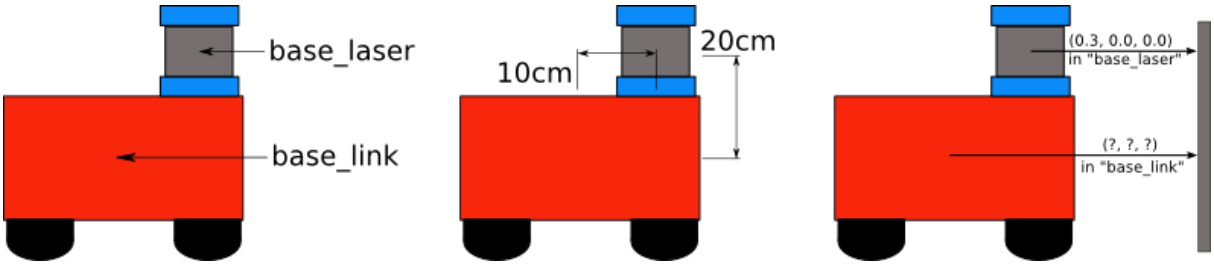
〈二〉、sensor_msgs/LaserScan：

本主題提供 RPlidar 測距雷達的數據提供給 move_base 實現即時定位與掃描突然出現的障礙物等功能。

〈三〉、tf：

RPlidar 接收到的消息是以 RPlidar 為坐標，但是導航時需要一個比較精確的小車坐標，此時必須將 RPlidar 的坐標轉換為小車底部的坐標。本主題的消息主要是處理不同坐標系間的轉換，我們令 RPlidar 的坐標為 base_laser，小車底部的坐標為 base_link，如下圖五所示。最後 tf 主題上的消息可提供給 move_base 作為即時定位與路線規劃時的參考。

參考網址：http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF



圖五、base_link 與 base_laser 之間的坐標轉換

〈四〉、odom：

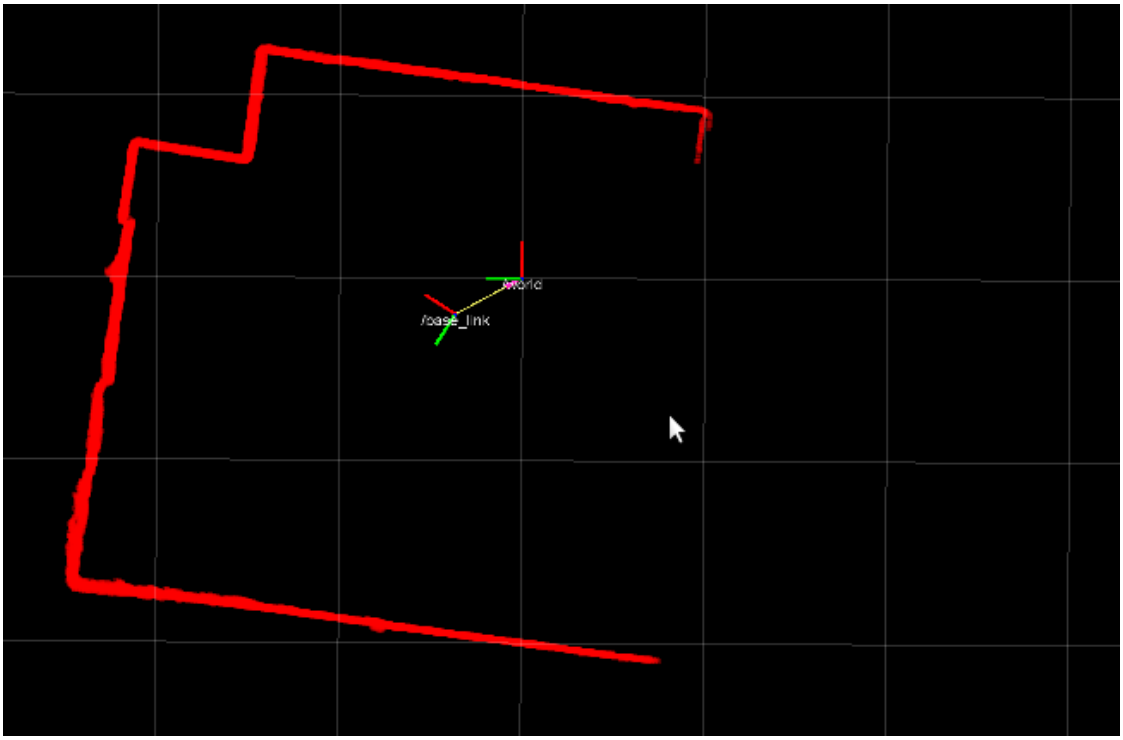
本主題是由機器人的里程計（Odometry）所發佈消息，由於小車原本未設計里程計，故本實驗是利用另一個開源 package，scan_tools 中的 laser_scan_matcher 來模擬里程計，利用 RPlidar 收集到的距離數據來模擬里程計。scan_tools 的安裝可以在終端機輸入指令：

sudo apt-get install ros-kinetic-scan-tools

laser_scan_matcher 利用 RPlidar 的雷達測距數據，模擬里程計，當雷達坐標移動時，另一個坐標會跟著移動，實際效果如下圖六所示。

odom 主題上的消息可提供給 move_base 作為機器人移動速度的運算與路線規劃速度的估計。

參考網址：http://wiki.ros.org/laser_scan_matcher



圖六、laser_scan_matcher 的模擬圖

〈五〉、cmd_vel：

本主題為 move_base 所發佈的消息，用來告訴小車現在該怎麼走，有 x 方向、y 方向、z 方向線速度，x 方向、y 方向、z 方向角速度，而本實驗關注的兩個參數為 x 方向線速度與 z 方向角速度。

關於 move_base 中的各個節點：

〈一〉、global_costmap 與 local_costmap：

此兩個 costmap 可以在小車行駛時，提供即時的建圖。當突然有障礙物或是人出現在小車周圍甚至是小車行駛的路徑上時，透過此兩個節點可以實現即時建圖的功能。而使用此節點必須先調教參數，參考網址：http://wiki.ros.org/move_base。實際參數如下：

1. costmap_common_params.yaml：

```
robot_radius: 0.05

obstacle_layer:
  enabled: true
  max_obstacle_height: 0.6
  min_obstacle_height: 0.0
  obstacle_range: 2.0
  raytrace_range: 5.0
  inflation_radius: 0.25
```

```
  combination_method: 1
  observation_sources: laser_scan_sensor
  track_unknown_space: true
  laser_scan_sensor: {data_type: LaserScan, topic: /scan, marking: true, clearing:
true, expected_update_rate: 0}

track_unknown_space: true

inflation_layer:
    enabled:true
    cost_scaling_factor:    10.0    # exponential rate at which the obstacle cost
drops off (default: 10)
    inflation_radius:         0.08# max. distance from an obstacle at which costs are
incurred for planning paths.

static_layer:
    enabled: true
    map_topic:"/map"
```

2. global_costmap_params.yaml：

```
global_costmap:
     global_frame: /map
     robot_base_frame: /base_link
     update_frequency: 1.0
     publish_frequency: 0
     static_map: true
     rolling_window: false
     resolution: 0.05
     transform_tolerance: 1.0
     map_type: costmap

GlobalPlanner:
     allow_unknown: false
```

3. local_costmap_params.yaml：

```
local_costmap:
   global_frame: odom
   robot_base_frame: base_link
   update_frequency: 3.0
   publish_frequency: 2.0    # Hz, at which the costmap will publish visualization information
   static_map: true
   rolling_window: true # true means that the costmap will remain centered around the robot as the robot moves through the world

   #width,height,resolution of the costmap
   width: 6.0
   height: 6.0
   resolution: 0.05
```

4. base_local_planner_params：

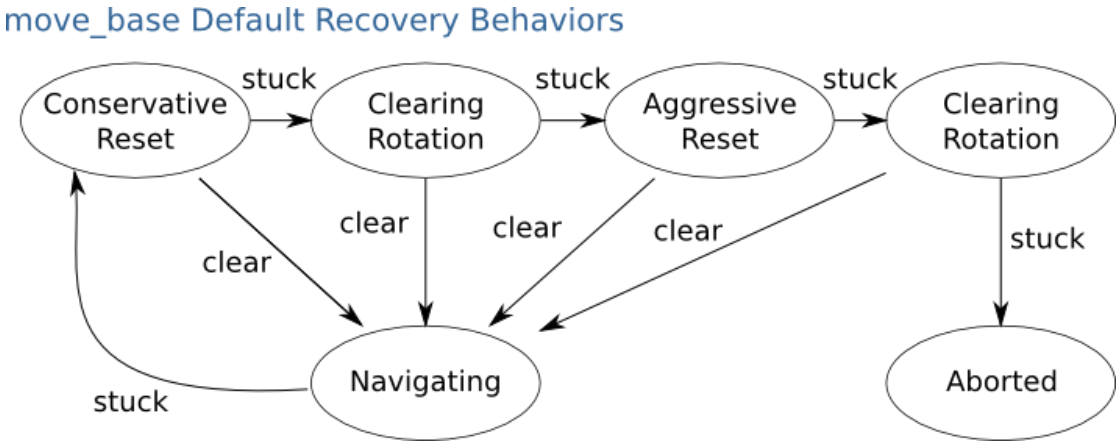
```
TrajectoryPlannerROS:
   max_vel_x: 0.45
   min_vel_x: 0.5
   max_vel_theta: 20.0
   min_in_place_vel_theta: 0.4

   acc_lim_theta: 3.2
   acc_lim_x: 2.5
   acc_lim_y: 2.5
   holonomic_robot: true
```

〈二〉、global_planner 與 local_planner：

此兩個節點為 Navigtion stack 重要的節點，其功能為判斷導航路線，global_planner 計算整條路線，local_planner 計算小車周圍的路線。在 Rviz 上，本實驗僅顯示 global_planner 所導航的路線。

〈三〉、recovery_behaviors：

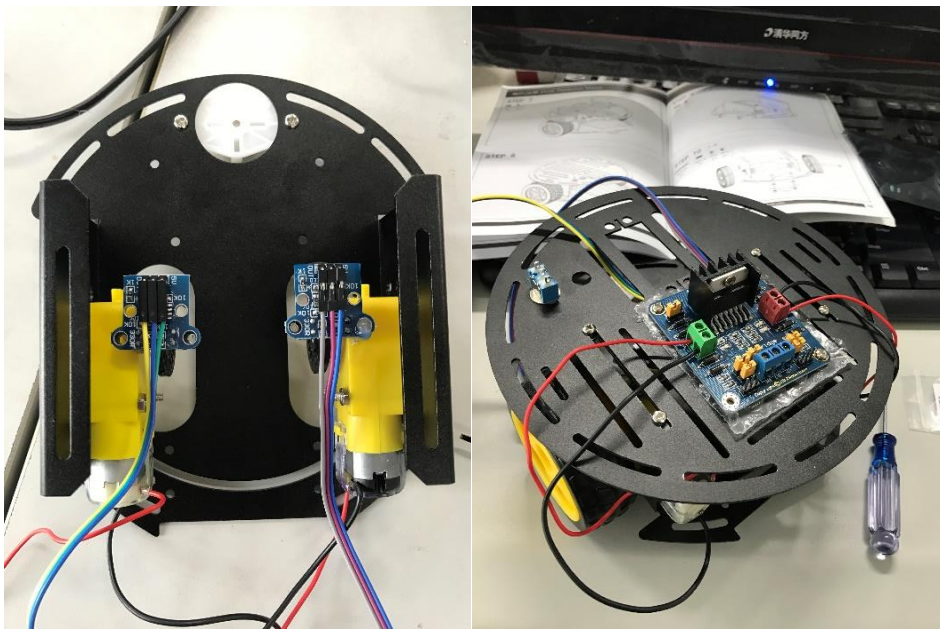
此節點用來即時修復小車導航的路線，下圖七為示意圖：



圖七、recovery_behaviors 示意圖

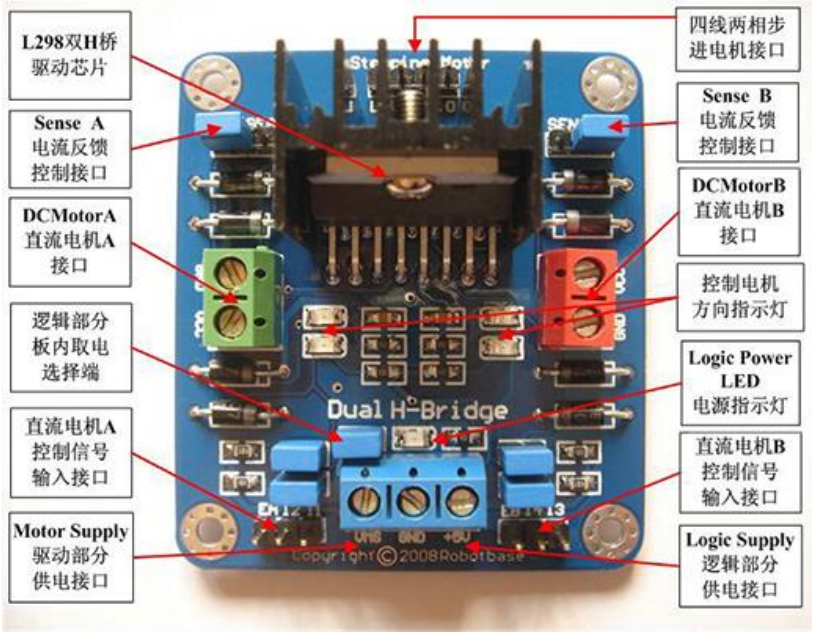
五、智慧小車的簡介與組裝

本實驗使用的是奧松機器人的智能小車套件，底部有兩個裝有光電碼盤的馬達與兩顆輪子，先將光電碼盤與馬達固定於底部，並為了方便辨識先把光電碼盤的接口記錄下來，完成後蓋上第二層板，如下圖八所示：



圖八、小車組裝過程

控制小車的驅動板是 L298N，如圖九所示：

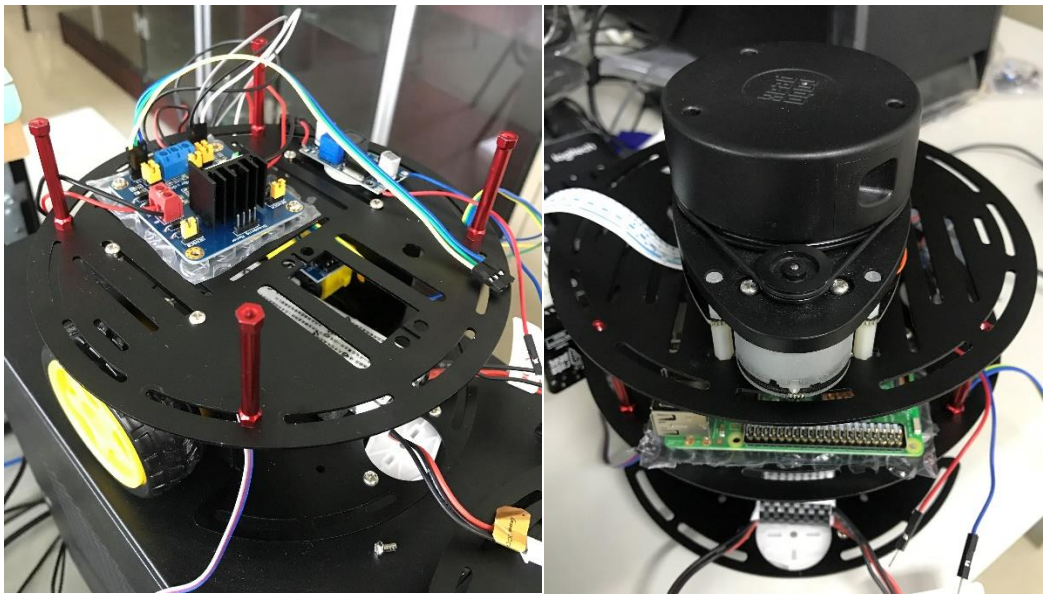


圖九、雙H橋驅動板 L298N

本實驗使用的開發板為 Raspberry Pi 3B ，如圖十所示：

圖十、Raspberry Pi 3B

為了將 RPlidar 固定在小車上，必須在加上第三層板子，首先先於第二層鎖上 4 組螺柱，而後將 RPlidar 固定在最上層，組裝過程如圖十一所示：



圖十一、小車組裝過程

六、實驗流程

〈一〉、利用 Hector SLAM 建圖

1. 使用 VNC viewer 用電腦遠端控制樹梅派，並使用鍵盤控制小車。（利用 pwm_keyboard.py）
2. 運行 Hector SLAM 後，緩慢地移動小車，盡量讓每個地圖上牆壁的黑線均勻。
3. 利用 map_server 將 Hector SLAM 建構好的地圖保存下來：
   rosrun map_server map_saver -f hectortest2.yaml
4. 保存完畢的圖如圖十二所示：

圖十二、521 實驗室 Hector SLAM 建圖

〈二〉、運行 Navigation stack 與 rviz：

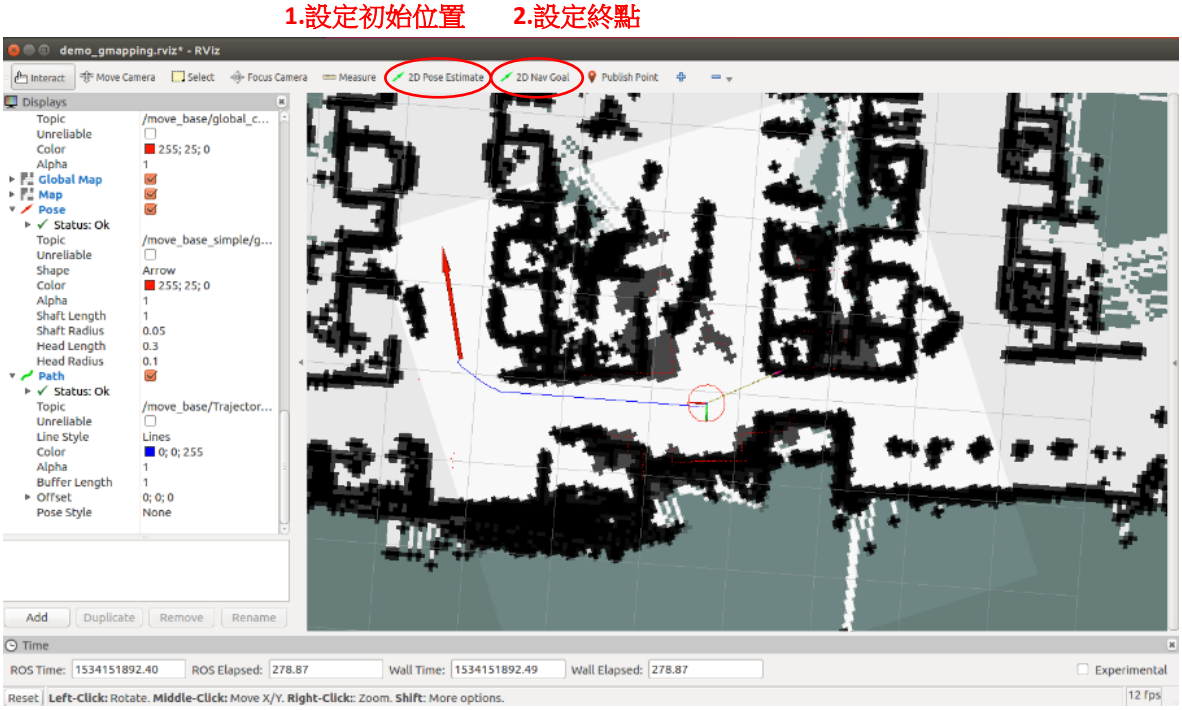
1. 利用 roslaunch 運行 Navigation stack 與 rviz。（demo3.launch）

```
<launch>
  <master auto="start"/>
  #### running the rplidar.launch ##################################
  <include file="$(find rplidar_ros)/launch/rplidar.launch" />


  #### publish an example base_link -> laser transform ################

  <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser"
    args="0.0 0.0 0.0 0.0 0.0 0.0 /base_link /laser 40" />

  #### start rviz ###################################################

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find laser_scan_matcher)/demo/demo_gmapping.rviz"/>

  #### start the laser scan_matcher #################################

  <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
    name="laser_scan_matcher_node" output="screen">

    <param name="fixed_frame" value = "odom"/>
    <param name="max_iterations" value="10"/>

  </node>

  #### run amcl#####################################################
    <include file="$(find amcl)/examples/amcl_omni.launch" />
  #### move_base ###################################################

   <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
    <rosparam file="/home/colin/test/costmap_common_params.yaml"
command="load" ns="global_costmap" />
    <rosparam file="/home/colin/test/costmap_common_params.yaml"
```

```
command="load" ns="local_costmap" />
      <rosparam file="/home/colin/test/local_costmap_params.yaml"
command="load" />
      <rosparam file="/home/colin/test/global_costmap_params.yaml"
command="load" />
      <rosparam file="/home/colin/test/base_local_planner_params.yaml"
command="load" />

   </node>
</launch>
```

2. 利用 map_server 將 521 實驗室的地圖傳給機器人：
   rosrun map_server map_server hectortest2.yaml

〈三〉、在 rviz 中上方設定初始位置與終點之後，自動生成導航路線：
效果為圖十一所示：

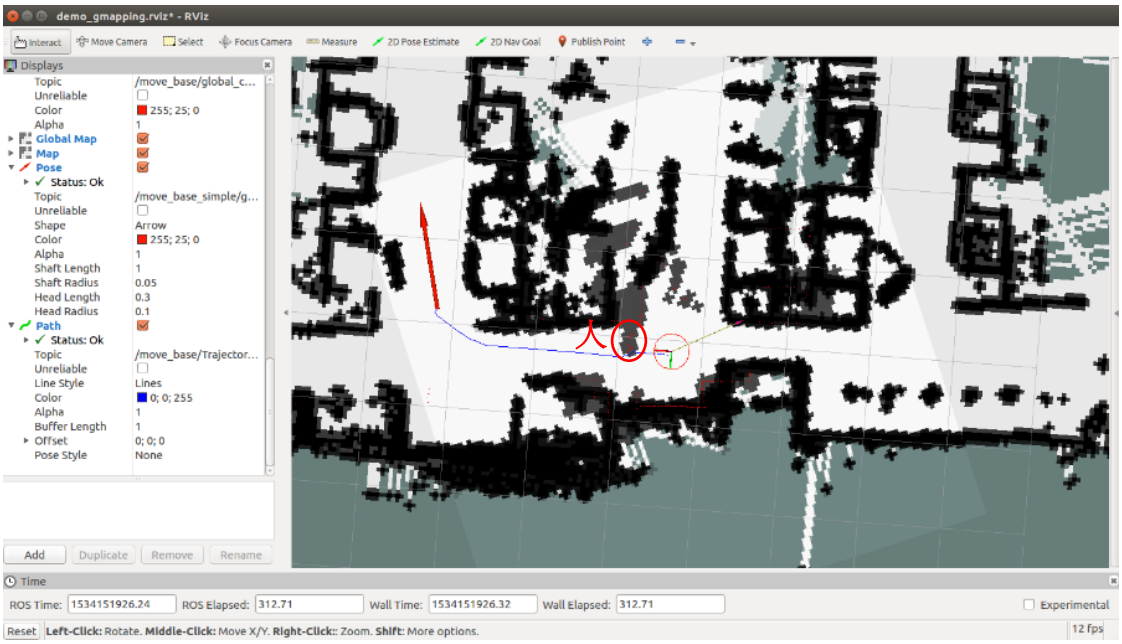


圖十一、設定初始位置與終點

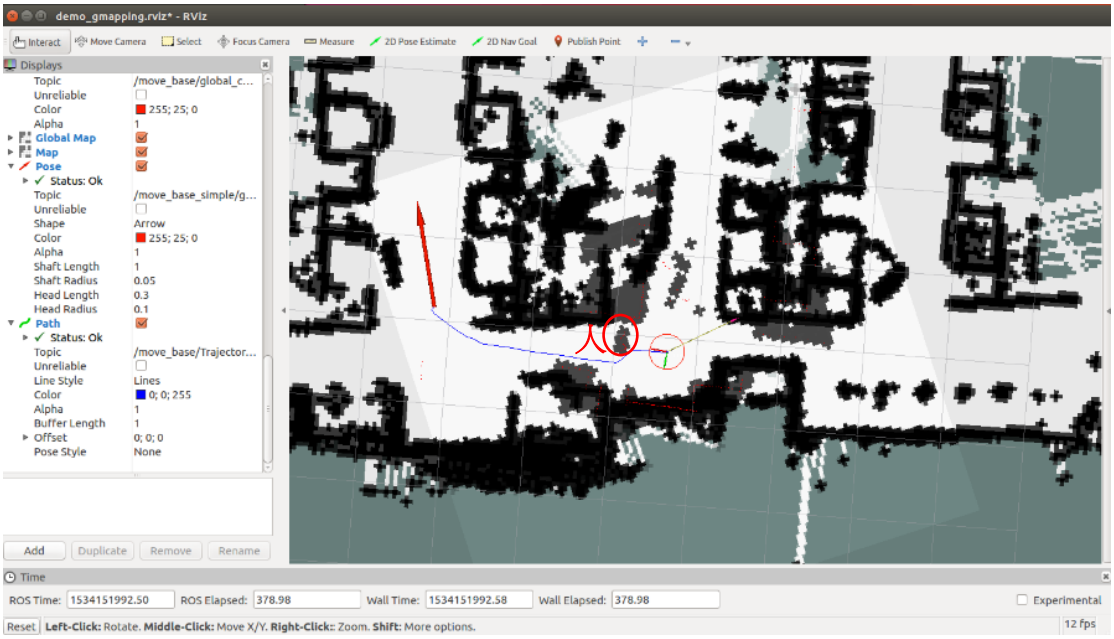
上圖十一中的紅色箭頭的尾端為設定之終點，藍線為導航的規劃路線，紅圈為機器人當前所在位置，仔細觀察可發現地圖中的障礙物線條較粗，這是因為 global_costmap 與 local_costmap 的效果，使得智慧小車能夠更加安全地感知距離與規劃路線。

如果突然有人或是物體阻擋在規劃的路線上，此時導航的規劃路線將會即

時更改，效果如圖十二、圖十三中的紅圈所示：



圖十二、有人阻擋時會改變導航路線



圖十三、有人阻擋時會改變導航路線

上圖十二、圖十三中，灰色的一塊阻擋在原本的導航路線代表的是人，由此可知智慧小車的地圖是即時更新的，並且會重新計算導航的路線規劃，讓智慧小車更順利地到達目的地。

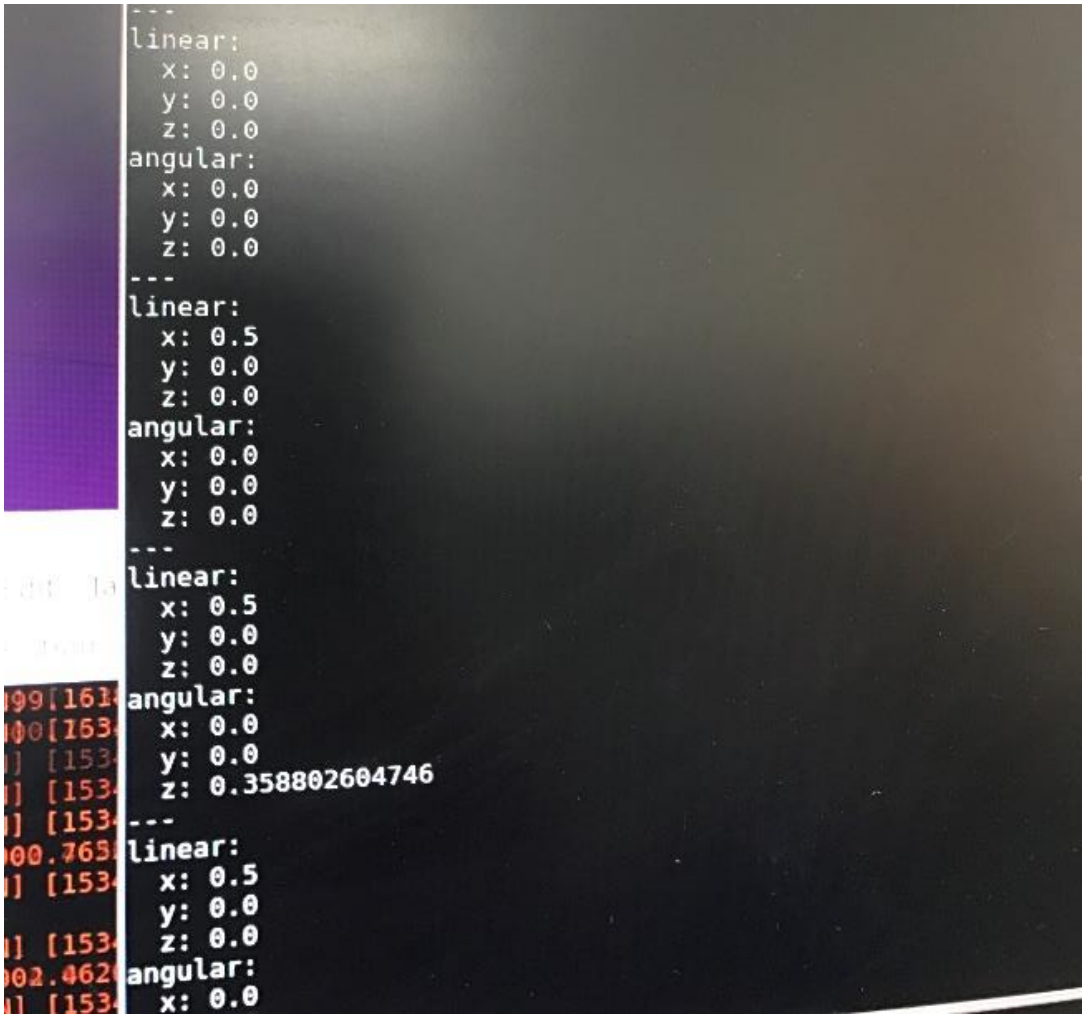
〈四〉、確認順利得到 cmd_vel 的消息：

運行指令：

rostopic echo /cmd_vel

即可看到 cmd_vel 上導航想要控制小車的消息，有 x 方向、y 方向、z 方向線速

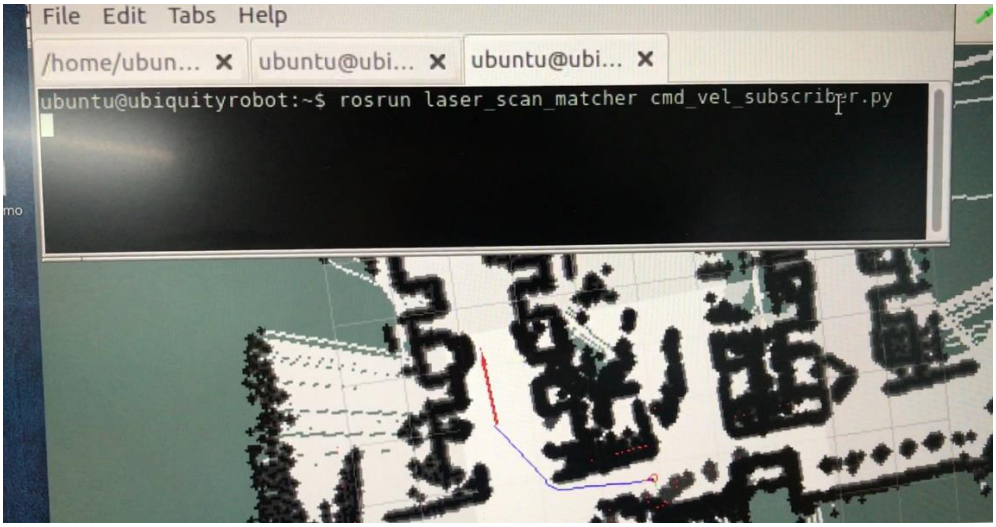
度，x 方向、y 方向、z 方向角速度，本實驗關注的兩個參數為 x 方向線速度與 z 方向角速度，如圖十四表示：



圖十四、cmd_vel 上的消息

〈五〉、運行接收 cmd_vel 消息進而控制小車的節點：

利用 rosrun 運行 cmd_vel_subsciber.py。



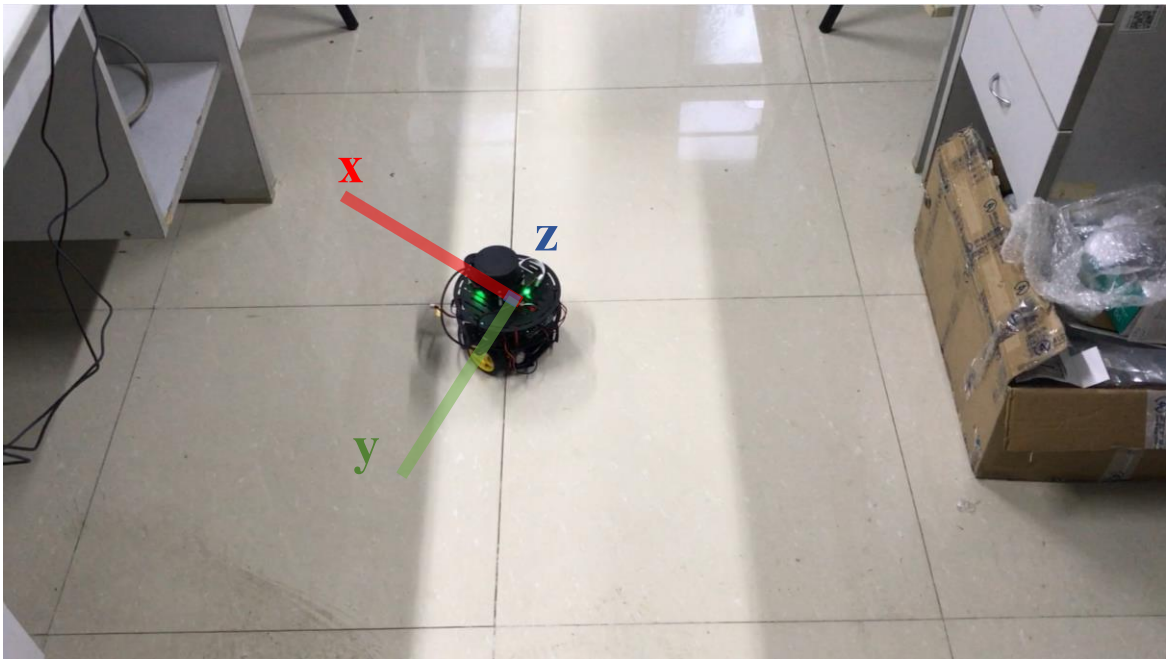
圖十五、運行接收 cmd_vel 消息進而控制小車的節點

〈六〉、觀察小車行駛情況：

圖十六、觀察小車行駛情況

〈七〉、比對小車位置與設定終點的一致性：

如下圖十七、圖十八



圖十七、小車終點位置

圖十八、設定終點之位置

由上圖十七與圖十八比對可知導航的結果順利，圖十八中小車位置與方向也和圖十七一致。