

第三章 進入 ROS

本章將說明 ROS 系統並透過遠端電腦可啟動 KUBOT 小車的相關功能。需要一台可遠端的 Linux 電腦、具有虛擬機並灌好 Linux 相關作業環境之電腦或是透過 Xshell 等軟體，可透過 SSH 連線進入 KUBOT 小車的樹梅派上位系統中。

在進入本章節前，建議對於 ROS 系統有基本的認識，可以先學習網路上基本的入門知識。
參考文獻裡有幾篇入門教學，可以先閱讀並嘗試操作。

事前準備：

1. 充好電的 KUBOT ROS 小車一台
2. Linux 環境之電腦一台 (需連接外網)

2-1 建立 SSH 連線

Setp.1 首先檢查是否安裝 ssh 軟件並啟用。在您的電腦呼叫終端機 (Ctrl+Alt+T)，輸入下方指令安裝 ssh，(此步驟的 sudo 密碼為您電腦的密碼，並且需連接外部網路)：

```
sudo apt - get install ssh
```

若未安裝系統會問是否安裝，輸入 y 後按 enter 就開始安裝了。

接著更新 ssh：

```
sudo apt - get upgrade
```

接著重新啟動 ssh：

```
sudo /etc/init.d/ssh restart
```

Setp.2 同第一章步驟 1-1，KUBOT 小車開機後，電腦透過 wi-fi 連線至 KUBOT 小車。

Setp.3 呼叫終端機 (Ctrl+Alt+T)，輸入下方指令進行連線 (@後的 IP 碼請依照出廠標籤設定)：

```
ssh kubot@192.168.172.1
```

Setp.4 若輸入無誤系統需要輸入密碼，密碼統一為 kubot。待系統連線後看到藍色標題即為成功。

備註：進入本系統後無法連接外網，除非額外擴充通訊模組。

樹莓派版本系統預設使用 oh-my-zsh shell。

並開機後自動啟動 rosluanch kubot_navigatiob gmapping_with_imu_with_cam.launch

建立主從關係後可以直接 rostopic list 差看。

2-2 關閉與啟動開機自啟動程序

```
cd /kubot2_ros
```

移除已啟動的 ros 節點

```
ps -aux | grep "/opt/ros" | grep -v "grep" | awk '{print $2}' | xargs kill -9  
ps -aux | grep "/piobot_ros/ros_ws" | grep -v "grep" | awk '{print $2}' | xargs kill -9
```

移除開機自啟

```
./kubot_remove_upstart.sh
```

啟動開機自啟 roslaunch

```
./kubot_add_upstart.sh
```

2-3 遠端鍵盤遙控模式

完成 2-1 與 2-2 後，可以嘗試使用鍵盤遙控 KUBOT 小車，為實現鍵盤遙控需要用 kubot_bringup 啟動下位機並另開終端機呼叫鍵盤遙控。

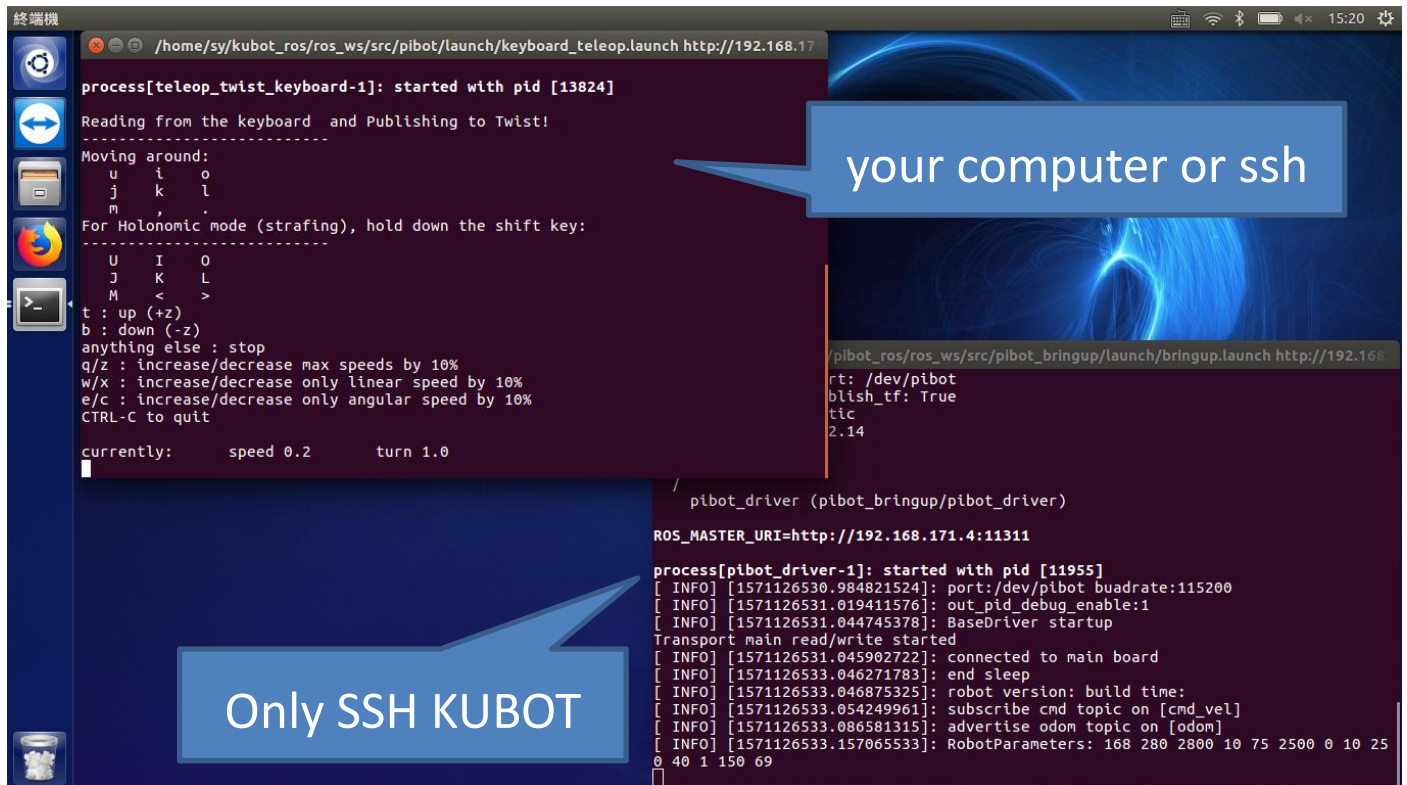
Setp.1 啟動 kubot 底層驅動，於終端機輸入：

```
kubot_bri
```

Setp.2 另開一個終端機，並按照 2-1 進入樹梅派系統當中，接著輸入：

kubot_key

接著就可以遙控小車，鍵盤指令如附圖。



```
終端機
/home/sy/kubot_ros/ros_ws/src/pibot/launch/keyboard_teleop.launch http://192.168.17
process[teleop_twist_keyboard-1]: started with pid [13824]
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
  u    i    o
  j    k    l
  m    ,    .
For Holonomic mode (strafing), hold down the shift key:
-----
  U    I    O
  J    K    L
  M    <    >
t : up (+z)
b : down (-z)
anything else : stop
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
CTRL-C to quit

currently:      speed 0.2      turn 1.0
█

pibot_ros/ros_ws/src/pibot_bringup/launch/bringup.launch http://192.168
rt: /dev/pibot
blish_tf: True
tic
2.14

pibot_driver (pibot_bringup/pibot_driver)
ROS_MASTER_URI=http://192.168.171.4:11311
process[pibot_driver-1]: started with pid [11955]
[ INFO] [1571126530.984821524]: port:/dev/pibot buadrate:115200
[ INFO] [1571126531.019411576]: out_pid_debug_enable:1
[ INFO] [1571126531.044745378]: BaseDriver startup
Transport main read/write started
[ INFO] [1571126531.045902722]: connected to main board
[ INFO] [1571126533.046271783]: end sleep
[ INFO] [1571126533.046875325]: robot version: build time:
[ INFO] [1571126533.054249961]: subscribe cmd topic on [cmd_vel]
[ INFO] [1571126533.086581315]: advertise odom topic on [odom]
[ INFO] [1571126533.157065533]: RobotParameters: 168 280 2800 10 75 2500 0 10 25
0 40 1 150 69
█
```

2-4 Gmapping 與 Navigation

roslaunch kubot_navigatiob gmapping.launch 預設會啟動機器人底盤(kubot_bri)、機器人激光雷達(kubot_lidar)、機器人模型(kubot_model)、move_base、gmapping 等功能，所以不需要事先啟動其他的檔案。如有啟動其他的執行檔請先關閉。

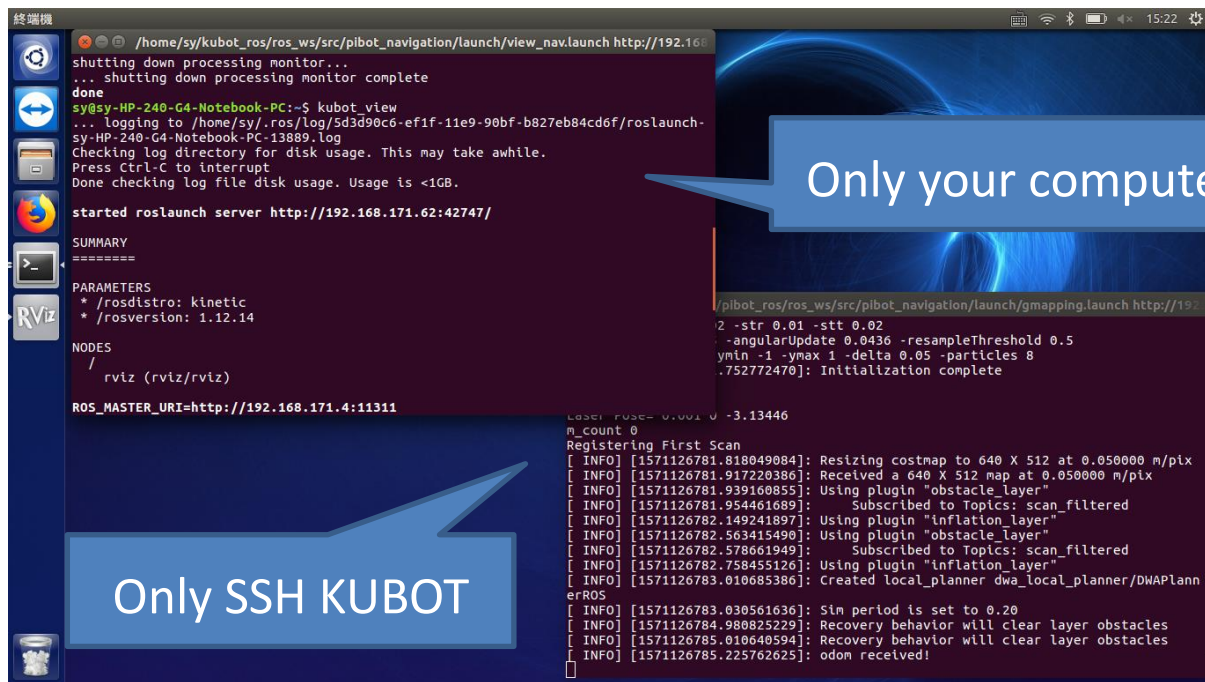
Setp.1 在車端(SSH 後的終端機)啟動建圖：

kubot_gmp

同樣，看到 odom received 即成功。

KUBOT

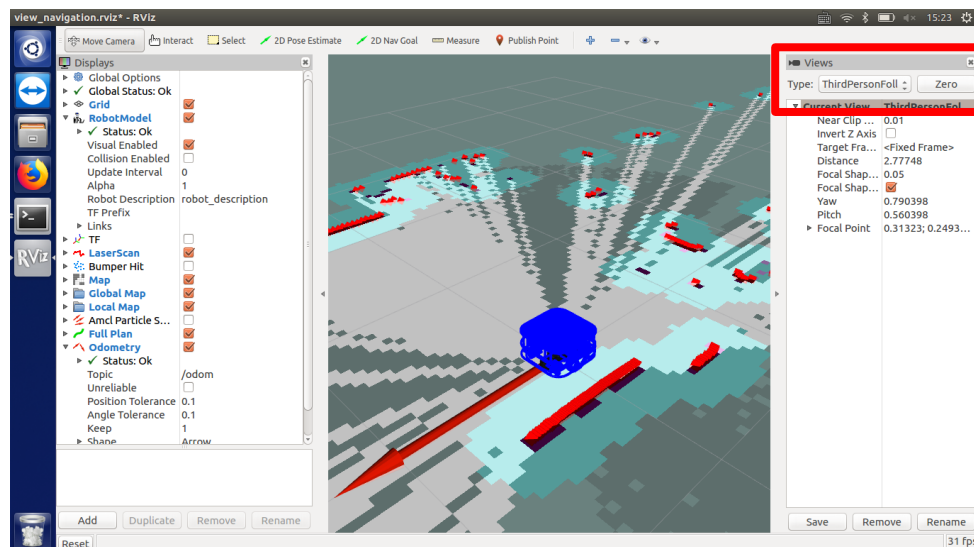
用戶手冊



Setp.2 接著在遠端電腦開一個終端機啟動 RViz：

kubot_view

就可以看到下面畫面。藍色小方塊是 KUBOT 小車 3D 模型，紅色框線為 lidar 判斷之邊界。黑色為已建立之牆(也可能是雜訊或噪點)，小車在後續進行導航時會判斷為需要避開、不行走之路段。白色區域為可行走區域，可透過導航指令讓小車自動前往，灰色為未知區域。



接著就可以按照步驟 2-3 另外啟動遙控程式，讓遙控小車去建立地圖了。

建圖時如欲外力(碰撞、擱淺等)使里程計計算錯誤，導致建圖破圖、重疊等問題，請重新開啟 gmp。此時亦可以透過 2D poseit 指定目標點與目標姿態讓小車做導航前往。這之間會邊導航邊建圖。若 odom 過於容易破圖，可以使用 kubot_gmp_imu，這是加入 ekf 融合 IMU 姿態迴授的建圖。(部分版本不支援)。

2-5 保存地圖 與 Navigation

如有建立好確定的地圖，下次不想重新建立，可以將地圖保存下來，下次打開導航就可以。

延續 2-4 Gmapping 的步驟，不要關掉 RViz 與 kubot_gmp 的 launch：

Setp.1 在車端再開一個終端機(步驟 2-1)並啟動存圖：

```
kubot_save_map
```

這個腳本內建將地圖存為 001，可以在 map 資料夾中看到名稱 001 柵欄式圖片。若想更換地圖代號，

可更換後方代碼，輸入：

```
kubot_save_map map_name := 001
```

存好地圖後可以將 Gmapping 和 RViz 關閉。(或是自行輸入地圖編號)

地圖預設保存於 kubot2_ros/ros_ws/src/kubot_navigation/maps 該資料夾當中。

Setp.2 接著啟動導航並調用編號 001 地圖。

```
kubot_nav
```

若沒有，則輸入：

```
kubot_nav map_name := 001.yaml
```

Setp.3 並在遠端電腦啟動 RViz：

```
kubot_view
```

此時小車和地圖不一定匹配，建議可以將小車搬回建圖時的起點再啟動 Nav，或是在 RViz 當中使用 2D pose estimate 設定小車現在在地圖當中的位置與姿態。然後同樣用 2D point 設定目標。

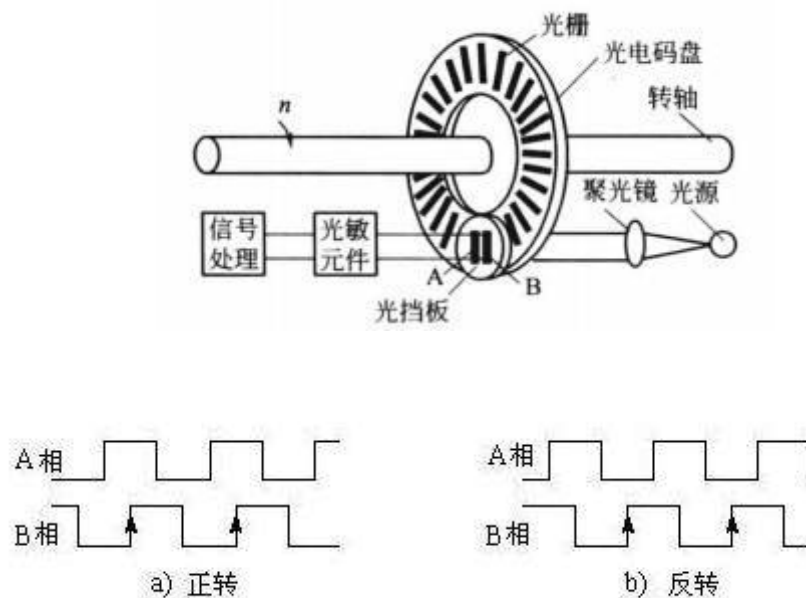
開發與學習筆記

本章將說明一些開發原理與參數的設置，讀者可以嘗試調整最佳化等，或是參考其他資源進行修改。

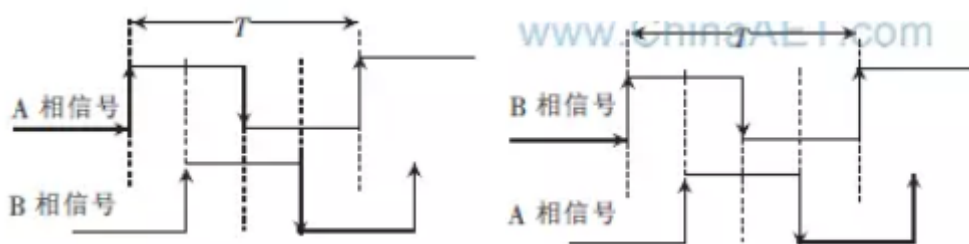
Encoder 與 PID 控制

對於 Robot 要實現位置與速度的控制需要透過感測器去獲取運動的資訊，而編碼器(Encoder)是很常用的原件。編碼器分為旋轉編碼器與線性編碼器，對於馬達這類的旋轉致動器，對應使用旋轉編碼器，而旋轉編碼器主要又分為絕對式編碼器與增量式編碼器。

在本產品當中是使用增量式的編碼器，安裝在直流有刷馬達後方，與後凸旋轉軸配合。對於旋轉軸美轉動一圈，增量式編碼器提供一定數量的脈衝訊號(通常編碼器在挑選時用 n 線/ n 脈衝表示)，通過量測脈衝的數量可以得到旋轉的角度，或是加上時間序列得到馬達轉軸的轉速。

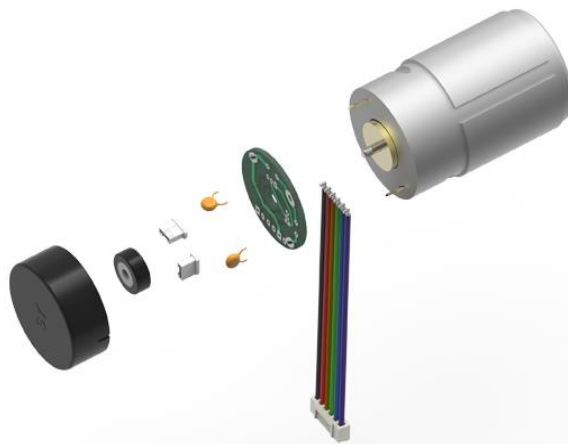


所謂四倍頻是指對 AB 相的上升與下降均做回授判斷，這樣在一個周期內會有四個狀態：



旋轉方向為正向時，A 相上升沿對應 B 相低電平-> B 相上升沿對應 A 相高電平-> A 相下降沿對應 B 相高電平-> B 相下降沿對應 A 相低電平。

旋轉方向為反向時·B相上升沿對應A相低電平->A相上升沿對應B相高電平->B相下降沿對應A相高電平->A相下降沿對應B相低電平



$$\text{Encoder resolution} = 4 (\text{Octave}) \times p \times R$$

$$p = \text{encoder 磁極數}$$

$$R = \text{齒輪箱/皮帶/鍊條 減速比}$$

Motor	CAP	Counts poles of per turn (PPR)		
		Current	limit.	
Ø 12	6.5	2, 6 (1, 3)	6 (3)	
Ø 15.4	6.5	2, 6 (1, 3)	6 (3)	
Ø 20.3	8.5	2, 6 (1, 3)	6 (3)	
Ø 30.0	12.6	2, 6, 14 (1, 3, 7)	14 (7)	
Ø 32	14.3	14 (7)	14 (7)	
Ø 36	13.5	14 (7)	14 (7)	
Ø 42.5	15.5	2, 10 (1, 5)	10 (5)	
Ø 52 & Ø 54	18.0	2, 10 (1, 5)	10 (5)	

Two Channel Encoder Connections

1. Black : - Motor
2. Red : + Motor
3. Brown : Hall Sensor Vcc
4. Green : Hall Sensor GND
5. Blue : Hall Sensor A Vout
6. Purple : Hall Sensor B Vout

One Channel Encoder Connections

1. Black : - Motor
2. Red : + Motor
3. Brown : Hall Sensor Vcc
4. Green : Hall Sensor GND
5. Blue : Hall Sensor A Vout
6. Purple : Hall Sensor B Vout