

Dongguan WHEELTEC INTELLIGENT technology co. , LTD

Aliexpress: minibalance.aliexpress.com/store/4455017

Taobao shop: minibalance.taobao.com

Web: www.wheeltec.net

ROS Development tutorial

Recommended to follow our official account for updated information



Release notes:

version	Date	Content description
V3.5	2021/03/09	First Release

The preface

The complete set of tutorials for ROS Navigator Car includes: "Moving Chassis Development Manual", "Ubuntu Configuration Tutorial", "ROS Development Manual". For the STM32 tutorial on Moving Chassis, see the documentation "Moving Chassis Development Manual"; For tutorials on how to build a development environment of Raspberry Pi and virtual machine , see "Ubuntu Configuration Tutorial".

The content of this document is mainly used to explain the development of ROS navigation robot, such as some preparatory work, how to build a map, how to achieve navigation and so on.

Two Ubuntu systems will appear in the following content. Please make a distinction between Ubuntu on Raspberry Pi and Ubuntu on a virtual machine. For more information about the user names and passwords of these two systems, see the following table:

Table 0-0

	User name and host name	The login password	The name of the WiFi	WiFi password	Static IP
Raspberry pie (Jetson Nano)	wheeltec	dongguan	See Table 0-1	dongguan	192.168.0.100
The virtual machine	passoni	raspberry	none	none	Custom configuration

Table 0-1

Car models	WIFI name
Mecanum wheel series	WHEELTEC66
Omnidirectional wheel series	WHEELTEC66
Ackermann series	WHEELTEC77
Differential series	WHEELTEC88

Directory

The preface.....	2
1. Fix Raspberry Pi peripheral serial port number.....	5
2. SLAM car ROS source code analysis.....	8
2.1 File system preview.....	8
2.2 Code composition.....	9
2.3 Serial communication with the lower computer.....	11
2.4 ROS topics and sensor data release.....	14
2.5 Robot node analysis.....	18
2.6 Parameter analysis of robot.....	20
2.7 Analysis of robot TF coordinate transformation.....	21
2.8 Start the robot through the launch file.....	24
3. Laser radar mapping.....	26
3.1 Start the mapping node.....	26
① gmapping.....	27
② hector and karto.....	28
3.2 Map preservation.....	29
4. Robot navigation.....	31
4.1 Start the navigation node.....	31
4.2 rviz navigation goal setting.....	32
4.3 Multi-point navigation.....	33
4.4 Navigation parameter setting.....	34
4.5 Navigation status monitoring and custom goals.....	38
4.6 Common navigation fault troubleshooting.....	39

1. Fix Raspberry Pi peripheral serial port number

When we use Windows system to connect some serial peripherals, we can see COM ports as shown in Figure 1-1 and Figure 1-2 through serial debugging assistant or device manager. These interfaces are usually changing

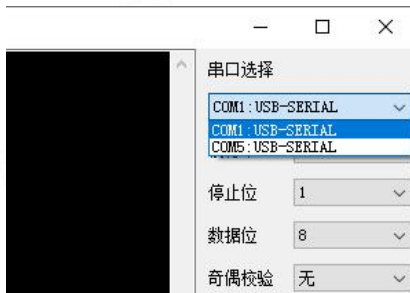


Figure.1-1 Serial port debugging assistant



Figure.1-2 Device manager

The programs on Raspberry Pi are usually run automatically by executing roslaunch from the command line or powering up, so it is not possible to select them manually every time. You need to know the corresponding ttyUsBX for each peripheral (x is 0-3). Taking lidar as an example, the original corresponding is ttyUSB2, but after re-plugging, the corresponding is ttyUSB0, as shown in Figure. 1-3 and 1-4.

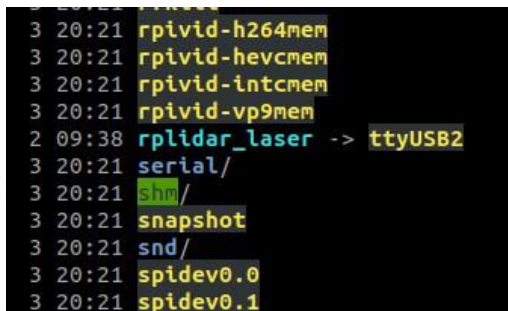


Figure.1-3 USB port for lidar

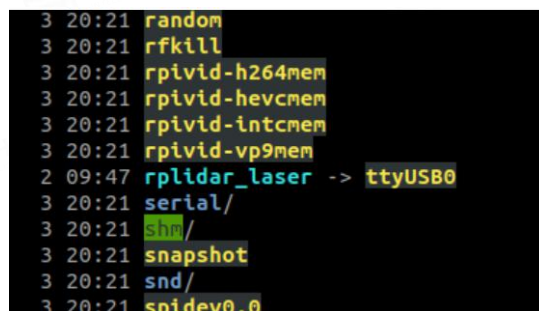


Figure.1-4 USB port after lidar re-plug

The above information is viewed by typing the command in the Raspberry Pi terminal, as shown in Figure 1-5:



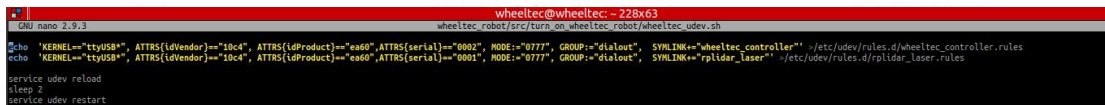
Figure.1-5 ll /dev

If the above problems are not solved, it will be very troublesome in the debugging process. We solve this problem by creating an alias for the USB device. Enter the instructions as shown in Figure 1-6 to open the following script file. If not, create it.

```
wheeltec@wheeltec:~$ nano wheeltec_robot/src/turn_on_wheeltec_robot/wheeltec_udev.sh
```

Figure.1-6 nano wheeltec_robot/src/turn_on_wheeltec_robot/wheeltec_udev.sh

After opening it, see Figure 1-7. Because this rule can not be wrapped, the code is too long and seems small. The text code is posted below:



```

wheeltec@wheeltec:~$ nano wheeltec_robot/src/turn_on_wheeltec_robot/wheeltec_udev.sh
wheeltec_robot/src/turn_on_wheeltec_robot/wheeltec_udev.sh
echo "KERNEL==\"ttyUSB*\", ATTRS{idVendor}==\"10c4\", ATTRS{idProduct}==\"ea60\",ATTRS{serial}==\"0002\",
MODE:=\"0777\", GROUP:=\"dialout\", SYMLINK+=\"wheeltec_controller\" >/etc/udev/rules.d/wheeltec_controller.rules
echo "KERNEL==\"ttyUSB*\", ATTRS{idVendor}==\"10c4\", ATTRS{idProduct}==\"ea60\",ATTRS{serial}==\"0001\",
MODE:=\"0777\", GROUP:=\"dialout\", SYMLINK+=\"rplidar_laser\" >/etc/udev/rules.d/rplidar_laser.rules
service udev reload
sleep 2
service udev restart
  
```

Figure.1-7 Scripts to modify rules

```

echo "KERNEL==\"ttyUSB*\", ATTRS{idVendor}==\"10c4\", ATTRS{idProduct}==\"ea60\",ATTRS{serial}==\"0002\",
MODE:=\"0777\", GROUP:=\"dialout\", SYMLINK+=\"wheeltec_controller\" >/etc/udev/rules.d/wheeltec_controller.rules
//The above two lines of code must be one line in the Nano text editor. No line breaks are allowed
echo "KERNEL==\"ttyUSB*\", ATTRS{idVendor}==\"10c4\", ATTRS{idProduct}==\"ea60\",ATTRS{serial}==\"0001\",
MODE:=\"0777\", GROUP:=\"dialout\", SYMLINK+=\"rplidar_laser\" >/etc/udev/rules.d/rplidar_laser.rules
//The above two lines of code must be one line in the Nano text editor. No line breaks are allowed
service udev reload
sleep 2
service udev restart
  
```

Among them, IDVendor and IDProduct are determined by USB to TTL chip, which will be checked by software below. ATTRS {serial} == "0002" is a serial number, this is the key, different modules are different, for example, here the CP2102 chip (USB cable) used by Raspberry Pi to connect to STM32 has serial port number 2, if change another CP2102 chip (USB cable) may be 1, therefore, it needs to be modified and fixed by the Windows software CP21XXCustomizationUtility, operation as shown in figure 1-8.

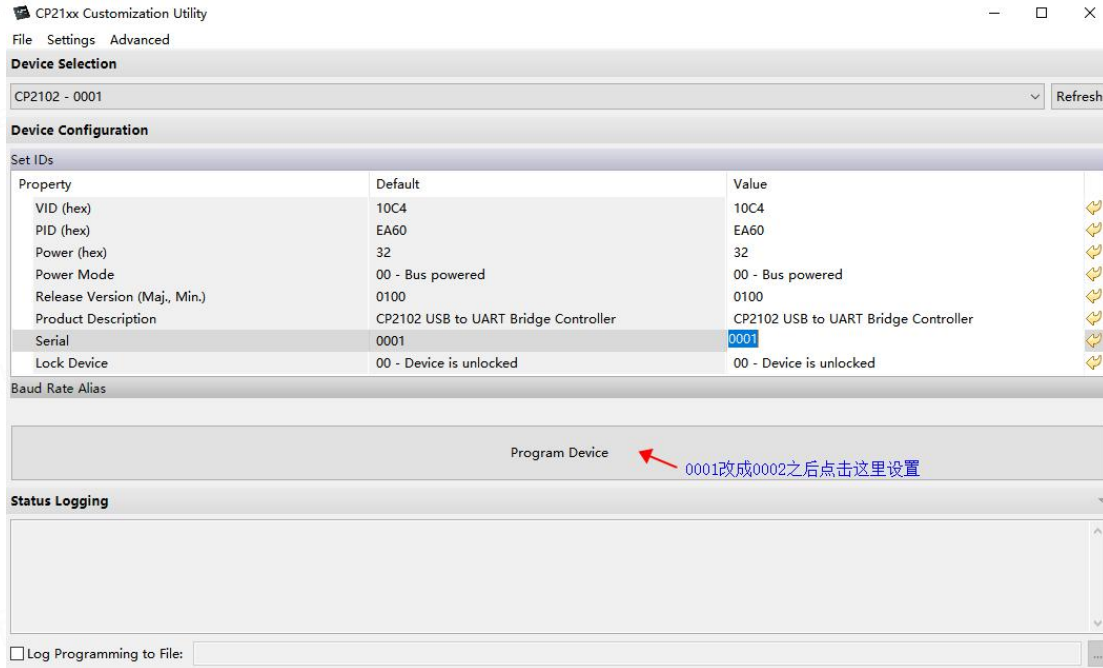


Figure 1-8 Permanently modify the CP2102 serial port number

After the above modification, the serial port number of CP2102 can be permanently fixed. Considering that the IDVendor and IDProduct of the same USB-TTL chip are also fixed, we can create aliases for USB peripherals with the above rule script. The above is just the file creation, we need to give permissions through the instructions in Figure 1-9:

```
wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot$ sudo chmod 777 wheeltec_udev.sh
```

Figure.1-9 sudo chmod 777 wheeltec_udev.sh

Then execute the script file, using the instructions in Figure 1-10:

```
wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot$ sudo ./wheeltec_udev.sh
```

Figure.1-10 sudo ./wheeltec_udev.sh

After that, the effect as shown in Figure 1-3 and 1-4 can be realized by re-plugging the USB peripheral. In the future, regardless of which USB port is connected to, we can use RPLIDAR_LASER to replace ttyUSBx when using radar, and we can use wheeltec_controller to replace ttyUSBX when connecting STM32 bottom layer.

2. SLAM car ROS source code analysis

In this chapter we discuss the ROS source code framework. Considering that the ROS basic tutorial is quite mature, especially the ROS official website is very detailed, this tutorial focuses on example applications, and is more based on the source code analysis of our robots. Considering that C++ is more efficient, this set of ROS robot Ubuntu source code is developed using C++.

2.1 File system preview

In the previous section, we learned how to use NFS mounts to make development more efficient and convenient. Enter the instructions in Figure 2-1-1 and Figure 2-1-2 to open the file system of the Raspberry Pi host in the virtual machine:

```
passoni@passoni:~$ sudo mount -t nfs 192.168.0.100:/home/wheeltec/wheeltec_robot /mnt
```

Figure 2-1-1 sudo mount -t nfs 192.168.0.100:/home/wheeltec/wheeltec_robot /mnt

```
passoni@passoni:~$ subl
```

Figure 2-1-2 subl

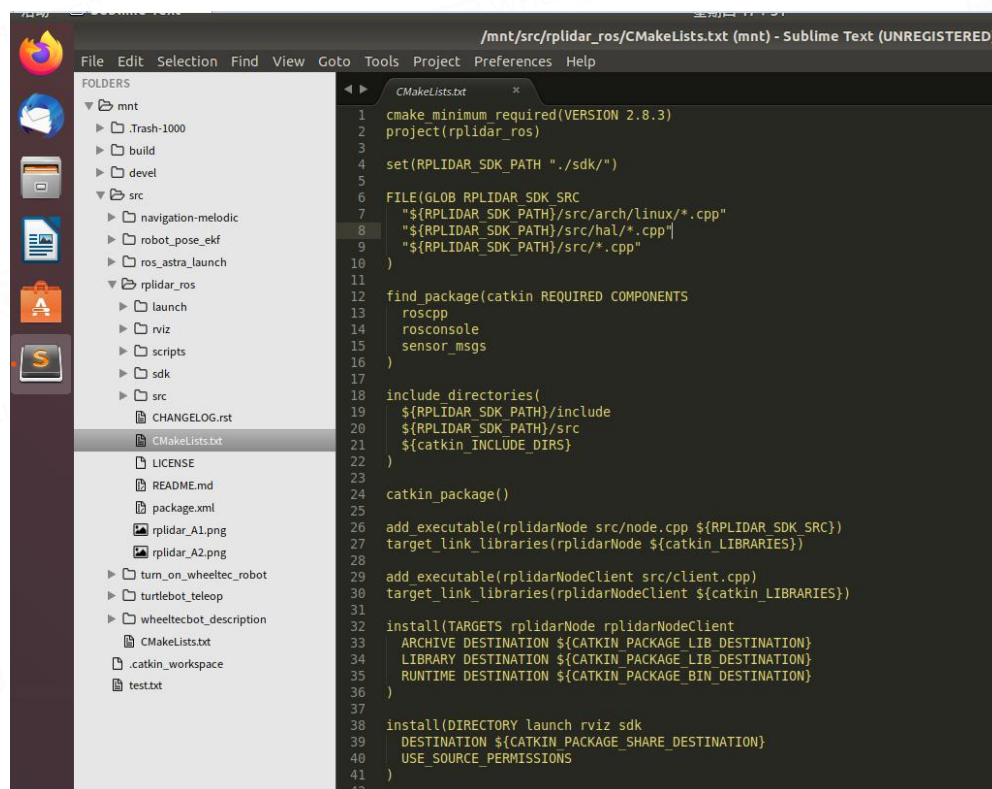


Figure.2-1-3 View the Raspberry Pi ROS source code in the virtual machine

The result is shown in Figure 2-1-3. This allows us to modify it as easily as we did with STM32. Save the modification and then go back to the terminal directory "~/wheeltec_robot" and compile with the command "catkin_make". During the development process, if you modify the launch launch file, YAML parameter file, etc., you can save it and run it again without compilation. To modify CPP files, CMakeLists compilation rules, etc., you need to execute the above compilation commands before running the program.

If you need to compile only one package, execute the instructions shown in Figure 2-1-4. This speeds up compilation, but then you just compile this package until you issue the instructions shown in Figure 2-1-5.

```
wheeltec@wheeltec:~/wheeltec_robot$ catkin_make -DCATKIN_WHITELIST_PACKAGES="turn_on_wheeltec_robot"
```

Figure 2-1-4 catkin_make -DCATKIN_WHITELIST_PACKAGES="turn_on_wheeltec_robot"

```
wheeltec@wheeltec:~/wheeltec_robot$ catkin_make -DCATKIN_WHITELIST_PACKAGES=""
```

Figure 2-1-5 catkin_make -DCATKIN_WHITELIST_PACKAGES=""

If you also need to speed up compilation, you can open multiple threads, such as catkin_make -j4, which is four threads. If the compile report Clock skew detected error, it is that the Linux system can only compile files whose file modification time is earlier than the current system time. You need to use the date -s command to set the time of Raspberry Pi to the current time, because Raspberry Pi can't connect to the Internet to update the time. The command is shown in Figure 2-1-6. Note that the current system time is used instead of the time to indicate the command.

```
wheeltec@wheeltec:~$ sudo date -s "2020-6-8 21:47:30"  
[sudo] password for wheeltec:  
2020年 06月 08日 星期一 21:47:30 UTC
```

Figure 2-1-6 sudo date -s "2020-6-8 21:47:30"

2.2 Code composition

From Figure 2-1-3, we can see that the ROS workspace consists of three folders, build, devel, and src, as well as the CMakeLists file.

src: Code space, where ROS packages are stored.

build: Compilation space, where intermediate files produced during compilation are stored.

devel: Development space, where the executable files generated by compilation, such as compiled nodes, are stored.

So we mainly look at the code in the src folder , inside the src are various metapackage and packages that implement functionality, package quantity depends on the complexity of the project, if you only need to run the slam A1 laser radar, that we only need to load the package "rplidar_ros", then run the launch file inside the package , this package can be downloaded in the website at the slam technology.

Similarly, when we are developing other hardware, we can also download the package from the corresponding official website. Most of the ROS-related codes can be downloaded from the ROS official website. Among them, the "navigation-melodic" navigation package in the ROS source code is a meta-function package "Metapackage", which also includes multiple packages, as shown in Figure 2-2-1 and 2-2-2.

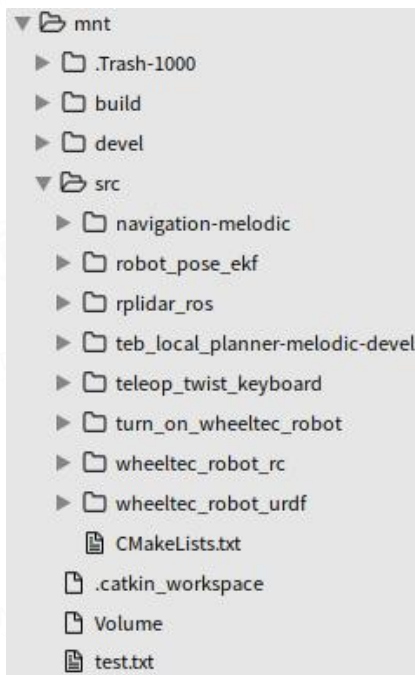


Figure.2-2-1 Each feature pack in SRC

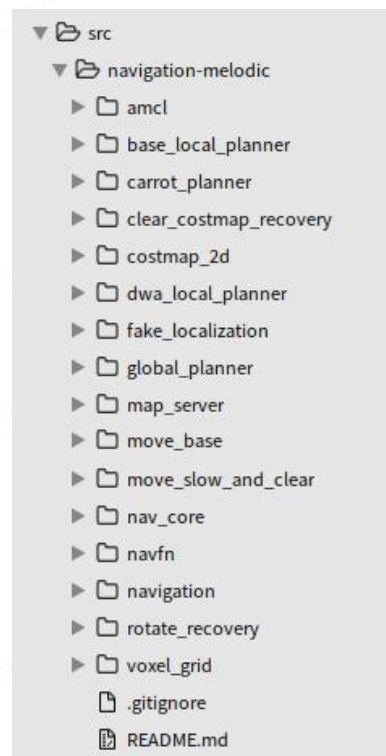


Figure.2-2-2 Navigation - Melodic meta feature pack

Among them, the function package "turn_on_wheeltec_robot" which

communicates with the underlying layer and publishes the sensor topics needed for navigation, mapping, etc., is the focus we need to understand. We noticed that there is a folder called `.trash-1000`. Ubuntu will create this folder when deleting a file, and if you accidentally delete a file, recent files will usually be restored from this folder.

2.3 Serial communication with the lower computer

The lidar we use communicates with the Raspberry Pi via a serial port, as does the STM32's underlying lower computer. The lidar is connected to the ROS system through the "rplidarNode" node of the official function pack, which will not be explained in more details here. The key point of development is the serial port communication between the STM32 underlying and the ROS system.

ROS is the operating system running on Raspberry Pi (Jetson Nano). Data such as IMU and odometer are needed for robot mapping and navigation. Theoretically, we can directly connect and collect these sensors through Raspberry Pi. Obviously, STM32 can do this very well, and STM32 has a hardware encoder interface. We can use STM32 to collect the information of encoder and IMU, and complete PID control. Then we can exchange sensor data and control instructions through serial port and Raspberry Pi.

Through the previous "STM32 Moving Chassis Development Manual", we have been familiar with the STM32 serial port related codes and communication protocols. The ROS side is highlighted here, and the code is shown in Figure 2-3-1.

```

Receive_Data.Frame_Header= Receive_Data.rx[0]; //The first part of the data is the frame header 0x7B //数据的第一位是帧头0x7B
Receive_Data.Frame_Tail= Receive_Data.rx[23]; //The last bit of data is frame tail 0x7D //数据的最后一位是帧尾0x7D

if (Receive_Data.Frame_Header == FRAME_HEADER) //Judge the frame header //判断帧头
{
    if (Receive_Data.Frame_Tail == FRAME_TAIL) //Judge the end of the frame //判断帧尾
    {
        if (Receive_Data.rx[22] == Check_Sum(22,READ_DATA_CHECK)||((Header_Pos==(1+Tail_Pos)))) //BBC check passes or two packets are interlaced //BBC校验通过或者两组数据包交错
        {
            Receive_Data.Flag_Stop=Receive_Data.rx[1]; //set aside //预留位
            Robot_Vel.X = Odom_Trans(Receive_Data.rx[2],Receive_Data.rx[3]); //Get the speed of the moving chassis in the X direction //获取运动底盘X方向速度
            Robot_Vel.Y = Odom_Trans(Receive_Data.rx[4],Receive_Data.rx[5]); //Get the speed of the moving chassis in the Y direction, The Y speed is only valid in the omnidirect
            Robot_Vel.Z = Odom_Trans(Receive_Data.rx[6],Receive_Data.rx[7]); //Get the speed of the moving chassis in the Z direction //获取运动底盘Z方向速度

            //MPU6050 stands for IMU only and does not refer to a specific model. It can be either MPU6050 or MPU9250
            //Mpu6050仅代表IMU, 不指代特定型号, 既可以是MPU6050也可以是MPU9250
            Mpu6050_Data.acele_x_data = IMU_Trans(Receive_Data.rx[8],Receive_Data.rx[9]); //Get the X-axis acceleration of the IMU //获取IMU的X轴加速度
            Mpu6050_Data.acele_y_data = IMU_Trans(Receive_Data.rx[10],Receive_Data.rx[11]); //Get the Y-axis acceleration of the IMU //获取IMU的Y轴加速度
            Mpu6050_Data.acele_z_data = IMU_Trans(Receive_Data.rx[12],Receive_Data.rx[13]); //Get the Z-axis acceleration of the IMU //获取IMU的Z轴加速度
            Mpu6050_Data.gyros_x_data = IMU_Trans(Receive_Data.rx[14],Receive_Data.rx[15]); //Get the X-axis angular velocity of the IMU //获取IMU的X轴角速度
            Mpu6050_Data.gyros_y_data = IMU_Trans(Receive_Data.rx[16],Receive_Data.rx[17]); //Get the Y-axis angular velocity of the IMU //获取IMU的Y轴角速度
            Mpu6050_Data.gyros_z_data = IMU_Trans(Receive_Data.rx[18],Receive_Data.rx[19]); //Get the Z-axis angular velocity of the IMU //获取IMU的Z轴角速度
            //Linear acceleration unit conversion is related to the range of IMU initialization of STM32, where the range is ±2g-19.6m/s²
            //线性加速度单位转化, 和STM32的IMU初始化的时候的量程有关, 这里量程±2g-19.6m/s²
            Mpu6050_linear_acceleration.x = Mpu6050_Data.acele_x_data / ACCEL_RATIO;
            Mpu6050_linear_acceleration.y = Mpu6050_Data.acele_y_data / ACCEL_RATIO;
            Mpu6050_linear_acceleration.z = Mpu6050_Data.acele_z_data / ACCEL_RATIO;
            //The gyroscope unit conversion is related to the range of STM32's IMU when initialized. Here, the range of IMU's gyroscope is ±500°/s
            //陀螺仪单位转化, 和STM32的IMU初始化的时候的量程有关, 这里IMU的陀螺仪的量程是±500°/s
            //因为机器人一般Z轴速度不快, 降低量程可以提高精度
            Mpu6050_angular_velocity.x = Mpu6050_Data.gyros_x_data * GYROSCOPE_RATIO;
            Mpu6050_angular_velocity.y = Mpu6050_Data.gyros_y_data * GYROSCOPE_RATIO;
            Mpu6050_angular_velocity.z = Mpu6050_Data.gyros_z_data * GYROSCOPE_RATIO;

            //Get the battery voltage
            //获取电池电压
            transition_16 = 0;
            transition_16 |= Receive_Data.rx[20]<<8;
            transition_16 |= Receive_Data.rx[21];
            Power_voltage = transition_16/1000+(transition_16 % 1000)*0.001; //Unit conversion millivolt(mv)->volt(v) //单位转换毫伏(mv)->伏(v)

            return true;
        }
    }
}
return false;

```

Figure.2-3-1 ROS system serial port receiving source code

According to the serial port communication protocol, the frame head is compared after receiving the data, and if the data match, the frame tail is verified. The idea of verification is that all the data of the packet except the frame tail are bit XOR, which can greatly improve the reliability of communication compared with the fixed frame tail. The serial port data verification code is shown in Figure 2-3-2.

```

/*****
Date: January 28, 2021
Function: Serial port communication check function, packet n has a byte, the NTH -1 byte
Input parameter: Count_Number: Check the first few bytes of the packet
功能: 串口通讯校验函数, 数据包n有个字节, 第n-1个字节为校验位, 第n个字节位帧尾. 第1个字节到第n-2个
输入参数: Count_Number: 数据包前几个字节加入校验 mode: 对发送数据还是接收数据进行校验
*****/
unsigned char turn_on_robot::Check_Sum(unsigned char Count_Number,unsigned char mode)
{
    unsigned char check_sum=0,k;

    if(mode==0) //Receive data mode //接收数据模式
    {
        for(k=0;k<Count_Number;k++)
        {
            check_sum=check_sum^Receive_Data.rx[k]; //By bit or by bit //按位异或
        }
    }
    if(mode==1) //Send data mode //发送数据模式
    {
        for(k=0;k<Count_Number;k++)
        {
            check_sum=check_sum^Send_Data.tx[k]; //By bit or by bit //按位异或
        }
    }
    return check_sum; //Returns the bitwise XOR result //返回按位异或结果
}

```

Figure 2-3-2 Serial Communication Checkout Function

After the verification, we can read the data in turn. The serial port receive function in Figure 2-6 is special in the following two lines of code:

```
Robot_Vel.Z = Odom_Trans(Receive_Data.rx[6],Receive_Data.rx[7]);  
//Robot_Vel.Z = Mpu6050.angular_velocity.z;
```

The Z-axis rotation angular velocity can be calculated indirectly by the encoder or acquired directly by the gyroscope. The gyroscope will have zero drift when it is stationary, and the encoder may slip when it is high-speed. According to the evaluation of the effect of actual operation, the effect of both can be satisfactory when the navigation accuracy is not too high. The above code is selected to use the encoder for data.

The ROS system receives the data sent by the lower computer. At the same time, the ROS system also sends instructions to the lower computer to control the robot, mainly the motion speed of each axis, as shown in Figure. 2-3-3.

```

/*****
Date: January 28, 2021
Function: The speed topic subscription Callback function, according to the su
功能: 速度话题订阅回调函数Callback, 根据订阅的指令通过串口发指令控制下位机
*****/
void turn_on_robot::Cmd_Vel_Callback(const geometry_msgs::Twist &twist_aux)
{
    short transition; //intermediate variable //中间变量

    Send_Data.tx[0]=FRAME_HEADER; //frame head 0x7B //帧头0x7B
    Send_Data.tx[1] = 0; //set aside //预留位
    Send_Data.tx[2] = 0; //set aside //预留位

    //The target velocity of the X-axis of the robot
    //机器人x轴的目标线速度
    transition=0;
    transition = twist_aux.linear.x*1000; //将浮点数放大一千倍, 简化传输
    Send_Data.tx[4] = transition; //取数据的低8位
    Send_Data.tx[3] = transition>>8; //取数据的高8位

    //The target linear velocity of the robot on the Y-axis.Only wheat-wheel an
    //机器人y轴的目标线速度。只有麦轮、全向轮小车才可以直接进行y轴移动
    transition=0;
    transition = twist_aux.linear.y*1000;
    Send_Data.tx[6] = transition;
    Send_Data.tx[5] = transition>>8;

    //The target angular velocity of the robot's Z axis
    //机器人z轴的目标角速度
    transition=0;
    transition = twist_aux.angular.z*1000;
    Send_Data.tx[8] = transition;
    Send_Data.tx[7] = transition>>8;

    Send_Data.tx[9]=Check_Sum(9,SEND_DATA_CHECK); //For the BBC check bits, see
    Send_Data.tx[10]=FRAME_TAIL; //frame tail 0x7D //帧尾0x7D
    try
    {
        Stm32_Serial.write(Send_Data.tx,sizeof (Send_Data.tx)); //Sends data to t
    }
    catch (serial::IOException& e)
    {
        ROS_ERROR_STREAM("Unable to send data through serial port"); //If sending
    }
}

```

Figure.2-3-3 The ROS system sends instructions to the lower computer

2.4 ROS topics and sensor data release

To start this section, let's run the launch file for the robot, with the command shown in Figure 2-4-1.

```
wheeltec@wheeltec:~$ roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch
```

Figure.2-4-1 roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch

"roslaunch" is the ROS command, "turn_on_wheeltec_robot" is a function package, and "turn_on_wheeltec_robot.launch" is the launch file below the package. After running, if there is no error, the robot will start normally. This launch file only starts the relevant nodes for communication and information release between the robot and the chassis, but does not start the nodes for remote control, mapping and

navigation, so that we can understand the bottom layer more easily.

A node can publish messages for a given topic, or it can follow a topic and subscribe to a specific type of data. The ROS system provides the `rostopic` command-line tool for displaying debugging information about ROS topics, including publisher, subscriber, publication rate, and ROS messages. We are not going to talk much about the message here, but the message here can be understood as a data structure. It supports standard types such as Integers, Booleans, Floating-Point, etc. It also supports nested structures and arrays, which are similar to the structure of C language. You can also customize the message type of the topic.

Let's start by running the instructions shown in Figure 2-4-2 to see the current list of topic names on the system. Note that we checked the topic on the Raspberry Pi host through `rostopic` command on the virtual machine, because the framework of multi-machine communication has been built before, and the following commands such as `rviz` are also run on the virtual machine, which can alleviate the performance bottleneck of the host of Raspberry Pi (Jetson Nano).

```
passoni@passoni:~$ rostopic list
/PowerVoltage
/amcl_pose
/cmd_vel
/joint_states
/mobile_base/sensors/imu_data
/nodelet_manager/bond
/odom
/odom_combined
/robot_cmd_vel
/robot_pose_ekf/odom_combined
/rosout
/rosout_agg
/scan
/smoothed_cmd_vel
/tf
/tf_static
/velocity_smoother/parameter_descriptions
/velocity_smoother/parameter_updates
```

Figure.2-4-2 rostopic list

The underlying sensor data of the robot is mainly encoder odometer data, IMU data and power supply voltage data, the data of power supply voltage is not necessary for navigation, although the underlying message is sent, the ROS system can receive it without the need to send it as a node, but we are here, you can see this topic in the robot runtime, convenient when battery voltage is too low to close the system, as shown in figure 2-4-3.

```
passoni@passoni:~$ rostopic echo /PowerVoltage
```

Figure 2-4-3 rostopic echo /PowerVoltage

Next, we run the command shown in Figure 2-4-4 to view the odometer data

```
passoni@passoni:~$ rostopic echo /odom
```

Figure 2-4-4 rostopic echo /odom

You can see the details for this topic in Figure 2-4-5. At this point we might think, the bottom of the mileometer sent only two or three data, why is here so complicated?

```
header:
  seq: 12026
  stamp:
    secs: 1591088313
    nsecs: 654938632
  frame_id: "odom"
child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: 0.00223159444616
      y: -1.05899398291e-05
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.00666921434198
      w: 0.999977760543
  covariance: [1e-09, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001, 1e-09, 0.0, 0.0, 0.0, 0.0, 0.0,
1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-09]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.0
  covariance: [1e-09, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001, 1e-09, 0.0, 0.0, 0.0, 0.0, 0.0,
1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-09]
```

Figure.2-4-5 Speedometer information

Because ROS is an open source distributed operating system, considering cross-platform compatibility, the data type of the odometer is fixed by ROS system, we just fill it in. The position information is obtained by velocity integration, general robot is 2 degrees of freedom, or 3 degrees of freedom, two wheels of differential car is 2 degrees of freedom, omnidirectional car is 3 degrees of freedom, two wheels of differential car incoming data is the x direction of the linear velocity and angular velocity of z direction, omnidirectional mobile car incoming data include linear velocity of the y direction. Other data are the current time of the system, frame_id, covariance used, and so on. In addition, we use the command shown in Figure 2-4-6 to see the types of messages published by this topic. The rostopic echo command is used to view the details of the topic, the rostopic type command is used to view the message type of the topic, and the rosmmsg show command is used to view the data

structure of the message.

```
passoni@passoni:~$ rostopic type /odom
nav_msgs/Odometry
```

Figure.2-4-6 rostopic type /odom

View the structure of the "nav_msgs/Odometry" data with the command in Figure 2-4-7. You can see that the data structure is exactly the same as shown in Figure 2-4-5.

```
passoni@passoni:~$ rosmmsg show nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
      float64[36] covariance
  geometry_msgs/TwistWithCovariance twist
    geometry_msgs/Twist twist
      geometry_msgs/Vector3 linear
        float64 x
        float64 y
        float64 z
      geometry_msgs/Vector3 angular
        float64 x
        float64 y
        float64 z
      float64[36] covariance
```

Figure.2-4-7 rosmmsg show nav_msgs/Odometry

Next we look at the IMU data and run the commands shown in Figure 2-4-8.

```
passoni@passoni:~$ rostopic echo /mobile_base/sensors/imu_data
```

Figure.2-4-8 rostopic echo /mobile_base/sensors/imu_data

You can see the details for the topic "IMU" as shown in Figure 2-4-9.

```
header:
  seq: 39145
  stamp:
    secs: 1591089669
    nsecs: 590859194
  frame_id: "gyro_link"
orientation:
  x: 0.0119222607464
  y: 0.000297941936878
  z: 0.140133276582
  w: 0.988357186317
orientation_covariance: [1000000.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 1e-06]
angular_velocity:
  x: 0.000266440009
  y: -0.000266440009
  z: 0.000532880018
angular_velocity_covariance: [1000000.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 1e-06]
linear_acceleration:
  x: 0.00048828125
  y: 0.022705078125
  z: 1.03125
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Figure.2-4-9 Innovative marketing information

The information passed in here includes the triaxial angular velocity and triaxial acceleration, and the quaternion is obtained by calculation. You can refer to the

odometer operation to see the message data type of the topic, which will not be described here.

2.5 Robot node analysis

A Node is a Node of the ROS system. A Node is an executable file that communicates with other nodes through the ROS Master. A node can publish or receive a topic, and a node can provide or consume a service. Rosnode is a command-line tool that displays debugging information about ROS nodes, including publish, subscribe, and connect. We use the command shown in Figure 2-5-1 to look at the list of nodes in the system.

```
passoni@passoni:~$ rosnodetool list
/base_to_gyro
/base_to_laser
/base_to_link
/joint_state_publisher
/nodelet_manager
/robot_pose_ekf
/robot_state_publisher
/rosout
/rplidarNode
/velocity_smoother
/wheeltec_robot
```

Figure.2-5-1 rosnodetool list

In this way, we can only see the list of nodes, which is not intuitive. Next, we use the rqt tool, using the command shown in Figure 2-5-2.

```
passoni@passoni:~$ roslaunch rqt_graph rqt_graph
```

Figure.2-5-2 roslaunch rqt_graph rqt_graph

You can see the relationship between nodes in Figure 2-5-3.

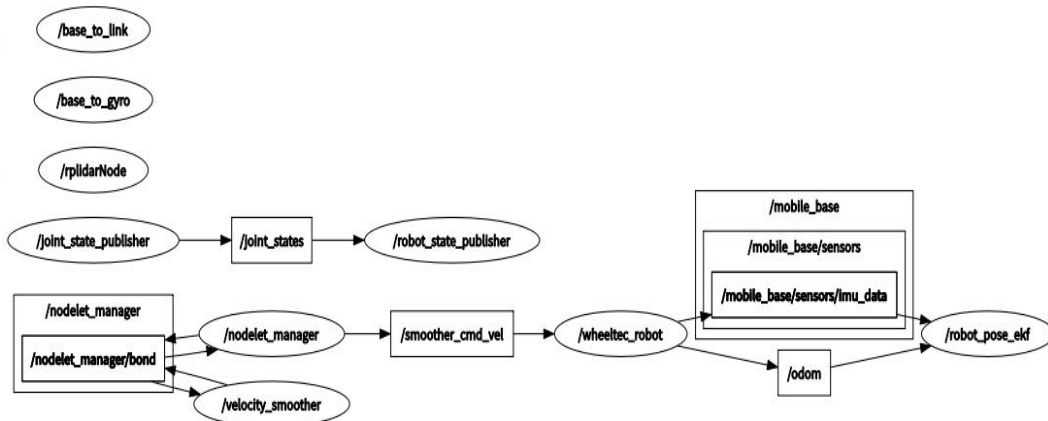


Figure.2-5-3 The list of nodes displayed by the RQT tool

The ellipse is the node name, and the box is the topic. The topic serves as a bridge between nodes. The underlying communication of ROS is based on "XML-RPC" protocol. It allows cross-platform software to make remote calls by sending and receiving messages in XML format, that is, it allows programs in different operating systems and environments to implement the specification and a series of methods based on the Internet procedure call.

Currently, only `turn_on_wheeltec_robot.launch` node related to the bottom layer of the robot has been opened, while nodes such as remote control, navigation and mapping have not been opened. As you can see from the list, nodes such as `/rplidarNode` are not used when navigation is not required. The `/joint_state_publisher` node describes the size of the robot and the position of the wheels. The `/nodelet_manager` node is provided by the ROS system.

Here's an example of `/wheeltec_robot` for a more detailed look at nodes. In the file `turn_on_wheeltec_robot.launch`, we can find the program that the node started, where `turn_on_wheeltec_robot` is the function pack where the node is located, `wheeltec_robot_node` is the executable file of the node, and `wheeltec_robot` is the name of the node after it is run, as shown in Figure 2-5-4.

```
<launch>
<!-- 打开节点wheeltec_robot, 初始化串口等操作 -->
<node pkg="turn_on_wheeltec_robot" type="wheeltec_robot_node" name="wheeltec_robot" output="screen">
  <param name="usart_port_name" type="string" value="/dev/wheeltec_controller"/>
  <param name="serial_baud_rate" type="int" value="115200"/>
  <param name="robot_frame_id" type="string" value="base_footprint"/>
  <param name="smoother_cmd_vel" type="string" value="smoother_cmd_vel"/>
  <param name="product_number" type="int" value="0"/>
</node>
</launch>
```

Figure.2-5-4 The wheeltec_robot node starts the code

We then use the command shown in Figure 2-5-5 to look at the details of this node.

```
passont@passont:~$ rosnode info /wheeltec_robot
-----
Node [/wheeltec_robot]
Publications:
* /PowerVoltage [std_msgs/Float32]
* /mobile_base/sensors/imu_data [sensor_msgs/Imu]
* /odom [nav_msgs/Odometry]
* /rosout [roscpp_msgs/Log]
* /tf [tf2_msgs/TFMessage]

Subscriptions:
* /amcl_pose [unknown type]
* /smoother_cmd_vel [geometry_msgs/Twist]

Services:
* /wheeltec_robot/get_loggers
* /wheeltec_robot/set_logger_level
```

Figure.2-5-5 rosnode info /wheeltec_robot

You can see that the /wheeltec_robot node has published five topics, subscribed to two topics, and provided two services. Two of the subscribed topics are provided with two callbacks in the wheeltec_robot.cpp program, as shown in Figure 2-5-6. The callback function here is similar to the external interrupt service function in SCM development. The external interrupt service function will enter after it is triggered and the callback function will execute after it receives a new message.

```
void turn_on_robot::Cmd_Amclvel_Callback(const geometry_msgs::PoseWithCovarianceStampedConstPtr &Pose)
{
    geometry_msgs::Pose Amclpose; //订阅机器人的姿态信息
    Amclpose.position.x = Pose->pose.pose.position.x;
    Amclpose.position.y = Pose->pose.pose.position.y;
    Amclpose.orientation = Pose->pose.pose.orientation;
    float temp = tf::getYaw(Amclpose.orientation);
}
```

Figure.2-5-6 Subscribe to the topic callback function

2.6 Parameter analysis of robot

The parameters in ROS system are similar to the global variables in SCM development and are managed by ROS Master. The communication mechanism is relatively simple and does not involve TCP/UDP communication. We can use the command "rosparm" to view or set the parameters of the robot, and execute the instructions shown in Figure 2-6-1 to view the current parameter list.

```
passoni@passoni:~$ rosparam list
/robot_description
/robot_pose_ekf/freq
/robot_pose_ekf/imu_used
/robot_pose_ekf/odometry_used
/robot_pose_ekf/sensor_timeout
/robot_pose_ekf/vo_used
/rosdistro
/roslaunch/uris/host_192_168_0_100_35079
/rosversion
/rplidarNode/angle_compensate
/rplidarNode/frame_id
/rplidarNode/inverted
/rplidarNode/serial_baudrate
/rplidarNode/serial_port
/run_id
/velocity_smoother/accel_lim_v
/velocity_smoother/accel_lim_w
/velocity_smoother/decel_factor
/velocity_smoother/frequency
/velocity_smoother/robot_feedback
/velocity_smoother/speed_lim_v
/velocity_smoother/speed_lim_w
/wheeltec_robot/robot_frame_id
/wheeltec_robot/serial_baud_rate
/wheeltec_robot/smooth_cmd_vel
/wheeltec_robot/usart_port_name
```

Figure.2-6-1 rosparam list

As an example, the `/wheeltec_robot/serial_baud_rate` parameter is set from the launch file, and some parameters can also be set from the YAML file. Note that what you see here are the parameters that are reported to the parameter server. Unreported local variables in the CPP file are not shown. You can view the value of this parameter by executing the instructions shown in Figure 2-6-2.

```
passoni@passoni:~$ rosparam get /wheeltec_robot/serial_baud_rate
115200
```

Figure.2-6-2 rosparam get /wheeltec_robot/serial_baud_rate

This is exactly the same as the parameters we set in the launch file. If you need to modify, you can execute the commands shown in Figure 2-6-3.

```
passoni@passoni:~$ rosparam set /wheeltec_robot/serial_baud_rate 9600
passoni@passoni:~$ rosparam get /wheeltec_robot/serial_baud_rate
9600
```

Figure.2-6-3 rosparam set /wheeltec_robot/serial_baud_rate 9600

As you can see, when we set it and look at it, the parameters have changed. However, this change only works for as long as the ROS is running. When we turn off the ROS system and turn it back on, the baud rate goes back to 115200. So, if we need to change a parameter permanently, we need to do so by modifying the source code, which is a parameter in a file such as launch or YAML. We can view or modify other parameters through similar instructions.

2.7 Analysis of robot TF coordinate transformation

Robot systems usually have many 3D coordinate systems that change with time, such as world coordinate system, fundamental coordinate system, etc. TF tracks all these frames over time and is a package to deal with different coordinate systems of

the robot. Coordinates for different parts of the robot and the world are stored in a tree structure. TF can transform points and vectors between any two coordinate systems into each other.

For readers who transfer from SCM control to ROS, the TF provided by ROS can greatly deepen their understanding of robots. In order to facilitate the understanding of TF coordinate transformation, we need to use the rqt tool. Next, let's execute the instructions as shown in Figure 2-7-1 to view the TF tree of the robot.

```
passoni@passoni:~$ rosrn rqt_tf_tree rqt_tf_tree
```

Figure.2-7-1 rosrn rqt_tf_tree rqt_tf_tree

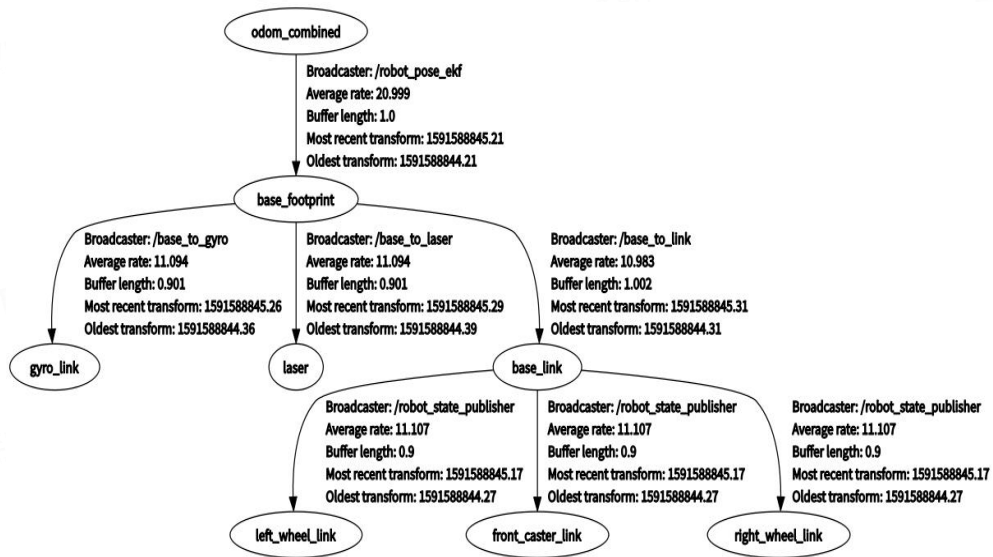


Figure.2-7-2 Robot TF tree

After opening, we can see the coordinate transformation relationship as shown in Figure 2-7-2. Robot_pose_ekf is an official ROS feature pack, and odom_combined is an IMU, odom_combined, and visual sensor coordinate system. The vision sensors can need not here. Base_link coincides with the robot center, the coordinate system origin can be the robot's rotation center, and the coordinate system origin of base_footprint is the projection of base_link origin on the ground. There is a slight difference between the two, and sometimes the Z value is different. It can be seen that the tf transformation of odom_combined->base_footprint is broadcast by robot_pose_ekf, that is, the deviation of the odometer position is obtained by EKF (Extended Kalman Filter).

As for the TF transformation between gyro_link, laser and base_link coordinate systems and base_footprint, we can take a look at the static coordinate transformation in the launch file as shown in Figure 2-7-3.

```

<!-- Arguments参数 -->
<arg name="car_mode" default="top_dff" doc="opt: top_dff, four_wheel_diff_bs,four_wheel_diff_dl"/>
<!-- 坐标变换, 需要实测 -->
<node pkg="tf" type="static_transform_publisher" name="base_to_link" args="0 0 0 0 0 base_footprint base_link 100" />
<node pkg="tf" type="static_transform_publisher" name="base_to_gyro" args="0 0 0 0 0 base_footprint gyro_link 100" />
<!-- car_mode and tf top_dff -->
<group if="$(eval car_mode == 'top_dff')">
  <node pkg="tf" type="static_transform_publisher" name="base_to_laser" args="0.101 0.00 0.107 3.1415 0 0 base_footprint laser 100" />
</group>

```

Figure.2-7-3 Robot static coordinate transformation

From the first and second nodes we can see that gyro_link, base_link and base_footprint coincide. From the third node we can see that the entry parameter is non-zero and the lidar is installed on the front of the car and base_footprint does not coincide. Args stand for x, y, z, yaw, pitch and roll in meters and radians, respectively. 100 represents one broadcast per 100ms. Figure 2-7-2 also shows the average frequency of broadcasts. The location of the radar installation may be different for each robot. So we use a custom parameter to select different models. To better understand the coordinates of the lidar, see Figure 2-7-4. The data in the figure is just an example. The specific size of each car needs to be measured.

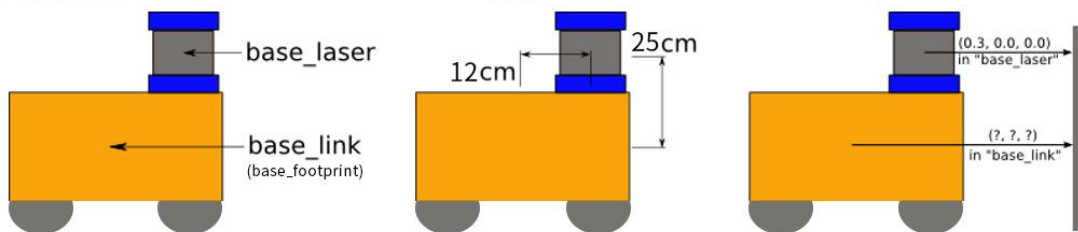


Figure.2-7-4 The relation between radar and robot coordinate system

The coordinate transformation of left_wheel_link, right_wheel_link and front_caster_link three coordinate systems to base_link is defined by wheeltec_robot.urdf unified robot description format file in wheeltec_robot_urdf function package. In order to more visually see the relationship between various coordinates, we input rviz on the virtual machine side, and each coordinate can be seen as shown in Figure. 2-7-5.

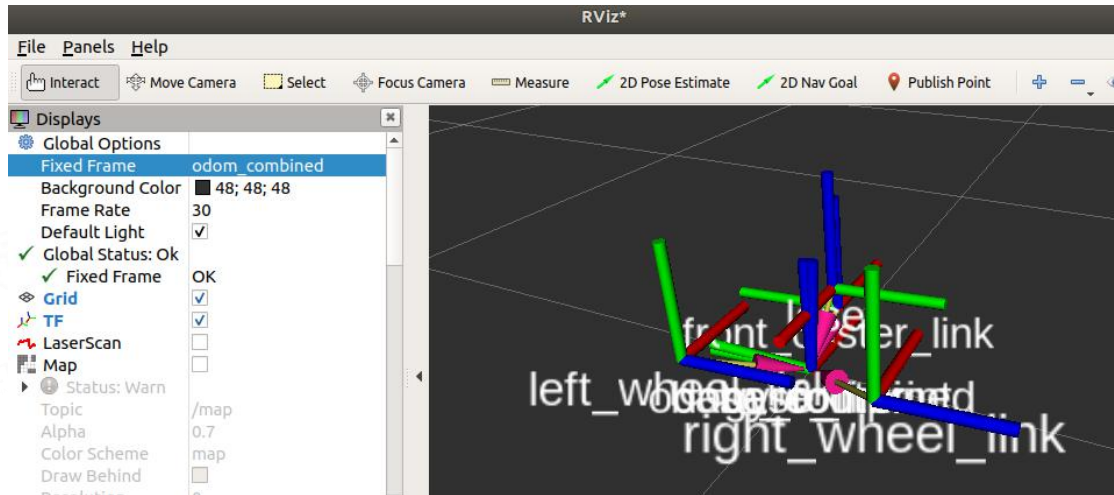


Figure.2-7-5 Coordinate diagram in RVIZ

Because of the RVIZ software, it looks a bit messy here, but you can still see the general coordinate relationship, such as laser_link. The red line is contrary to the others, indicating that the radar is installed by rotating 180° in our system. When we run it later, we'll see that odom_combined will inevitably drift randomly over time.

2.8 Start the robot through the launch file

After the ROS source code is written and compiled correctly, we can start. Rosrun command can be used to start ROS nodes, but robots generally consist of multiple nodes, so it is inefficient to start nodes one by one through rosrn, so we use roslaunch command to start launch file, which can start multiple nodes at the same time and can also set parameters. The list of launch files is shown in Figure 2-8-3. Here, the file name of the launch for the robot is turn_on_wheeltec_robot.launch. The remote robot needs to launch the keyboard_teleop.launch file under the wheeltec_robot_rc feature package. See at figure 2-8-1 and 2-8-2. After starting the robot, we can control the robot through the keyboard. When remote control, we need to ensure that the terminal of the starting this node is not in the background motion state. The robot remote control node may be used when we do the mapping later. Open the mapping.launch file if you need to start mapping, and open the navigation.launch file if you need navigation.

```
wheeltec@wheeltec:~$ roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch
```

Figure 2-8-1 roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch


```
wheeltec@wheeltec:~$ roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

Figure 2-8-2 roslaunch wheeltec_robot_rc keyboard_teleop.launch

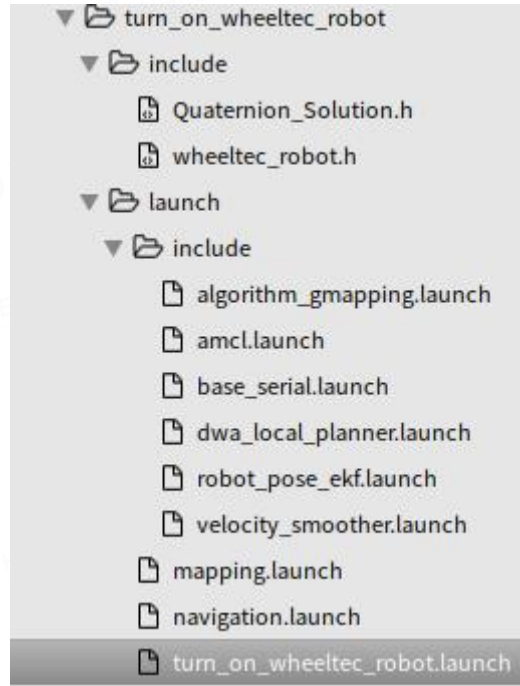


Figure 2-8-3 Launch file list

3. Laser radar mapping

The ROS open source community has collected a variety of SLAM algorithms, which can be directly used or redeveloped. Gmapping is a commonly used open source SLAM algorithm based on filtering SLAM framework, and it is also the most commonly used and mature functional package at present. In addition to gmapping, we also provide hector and karto mapping algorithms.

Gmapping can build indoor maps in real time, which requires less computation and high precision to build small scene maps. Compared with hector SLAM, which requires lower frequency of lidar and has higher robustness, it is easy for hector to mismatch when the robot quickly turns, resulting in dislocation of the built map. The main reason is that the optimization algorithm is easy to fall into the local minimum value. Gmapping, however, does not require many particles and does not have loop detection when compared to Cartographer for building small scene maps, and therefore does not require much less computation than Cartographer without much less accuracy. Gmapping makes good use of wheel odometer information, which is why it has a low lidar frequency requirement: the odometer provides a priori of the robot's position and pose. The karto mapping algorithm has the same principle as gmapping, which mainly relies on the odometer information to complete the mapping.

Hector and cartographer were not designed to address the location and mapping of planar mobile robots. Hector was designed to be used in disaster relief and other areas of rough terrain, and therefore odometers were not available. The cartographer was used for handheld lidar to complete the SLAM process, meaning that odometer information was not required at all.

3.1 Start the mapping node

Before running the mapping node, we first open the launch file of the mapping node to check the contents inside, can see the way choice of mapping, the default is to use gmapping way, we can also use a custom modified into karto and hector mapping, only need to modify the contents of the default = "" quotes after save, then re-run built

figure node.

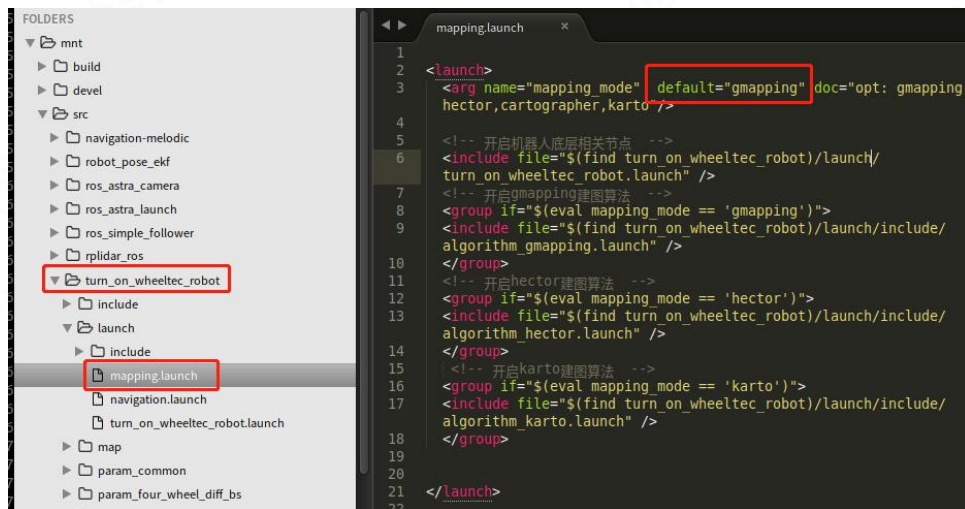


Figure.3-1-1 Mapping. Launch file

① gmapping

I won't go into much more detail here about how gmapping works. There is a lot of systematic information on the principle of gmapping, both in the paper and on the official website. Now let's go straight into the application and run the instructions shown in Figure 3-1-2.



Figure.3-1-2 roslaunch turn_on_wheeltec_robot mapping.launch

After running, we need to open rviz in the virtual machine. After adding the map through add in the lower left corner, we can see that the map is partially built in the right window, as shown in Figure 3-1-3.

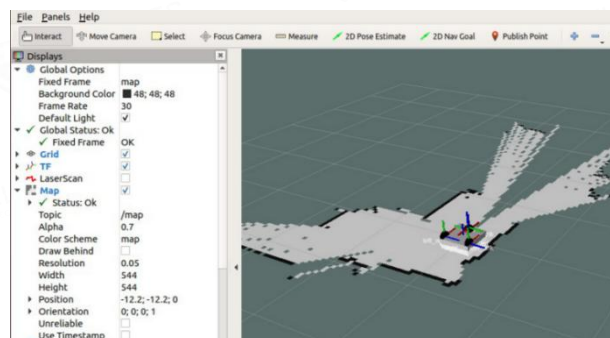


Figure.3-1-3 RVIZ opens the state of the map

Here are two techniques for laser radar mapping.

Tip 1: Small loop, then big loop. Try to control the robot to go through the small loop that can be closed quickly, and then gradually expand to the periphery to build

the map. Avoid direct attempt of large loop closed loop, if the cumulative error is large, it will lead to failure. Avoid taking a path unrelated to the current loop closed loop, which will generate cumulative errors in the process, and easily lead to too large a gap between the beginning and the end of the loop, resulting in the inability to close the loop.

Tip 2: Close the loop, then perfect the details. Avoid circling and walking back and forth in pursuit of building details before loop closure. This can easily lead to closed-loop failures or errors when the environment has fewer characteristics. If the map needs to be improved, the device should first walk in a straight line to quickly complete the closed loop, and then on the closed path, further scan the map to improve the details.

Some other tips can be summarized by yourself or refer to the advice provided by the lidar manufacturer. Here, the robot is controlled by APP(keyboard is also OK), and we have completed the creation of the map, as shown in Figure 3-1-4.

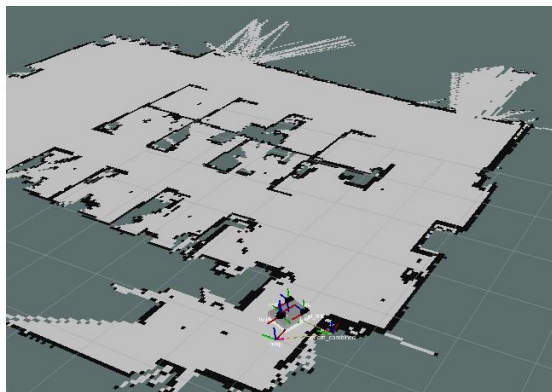


Figure.3-1-4 The state of completion of drawing

② hector and karto

The principle of how karto and hector built the map will not be described here. The mapping principle of karto is roughly the same as that of gmapping, but when the robot moves at a lower speed, the mapping effect is slightly better than that of gmapping algorithm. The hector mapping algorithm does not require the odometer information at all, so you can hold the robot in your hand to complete the mapping, but it is also possible to control the robot in space as well as the other two methods.

3.2 Map preservation

After the mapping in the previous section is completed, the previous windows cannot be closed. The current Ubuntu terminal window is shown in Figure 3-2-1.

Next, let's start the map_server node to save the map and enter the path "/home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/map" and execute the command as shown in Figure 3-2-2. So we can save the map, called mymap, and if we already have the mymap file, this command will overwrite the previous map. If mymap is removed from the previous instruction, the system will save map as the map name, and the map will be automatically saved in the directory where the current terminal is located. The map includes not only a mymap.pgm map data file, but also a mymap.yaml file, which is a configuration file about the map where mymap.pgm can also be edited using software such as GIMP.

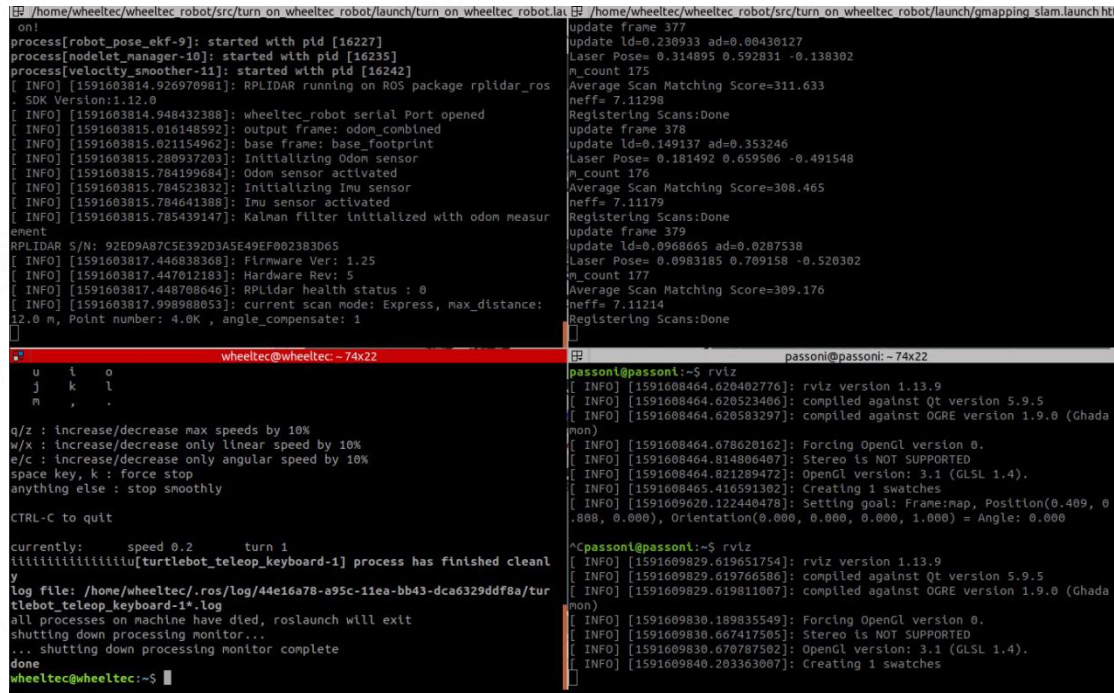


Figure.3-2-1 Ubuntu terminal window

```
wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot/map$ rosrn
map_server map_saver -f mymap
[ INFO] [1591606830.563462054]: Waiting for the map
[ INFO] [1591606830.818335257]: Received a 544 X 544 map @ 0.050 m/pix
[ INFO] [1591606830.818451128]: Writing map occupancy data to mymap.pgm
[ INFO] [1591606830.857631924]: Writing map occupancy data to mymap.yaml
[ INFO] [1591606830.858044405]: Done

wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot/map$ ls
map30.pgm  map40.pgm  maphds.pgm  map.pgm  mymap.pgm
map30.yaml  map40.yaml  maphds.yaml  map.yaml  mymap.yaml
```

Figure.3-2-2 rosrn map_server map_saver -f mymap

4. Robot navigation

The key to navigation is to realize robot positioning and path planning. ROS provides two function packages, `move_base` and `amcl`, as solutions. `Move_base` can realize optimal path planning in robot navigation, and `amcl` can realize robot positioning in two-dimensional map. We only need to set the target position and posture of the robot in the map, and the robot can arrive at the optimal path according to the map.

4.1 Start the navigation node

In the previous chapter, we have completed SLAM through the mapping algorithm and built a map based on the environment where the robot is located. Sometimes we will see a lot of maps in the map folder. Some maps are not built in the environment where the robot is located. For example, the maps built by the manufacturer during debugging cannot be used by the customer. Therefore, before launching the navigation node, we open the navigation launch file to make sure that the map selected by the navigation launch file is the previously established map `mymap`, as shown in Figure 4-1-1.

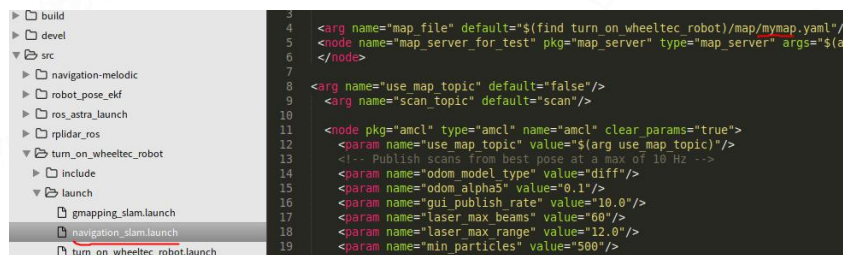


Figure.4-1-1 The navigation_slam.launch file has been modified

After modification, we save it, and then close all the previous nodes, and place the robot at the starting point of mapping. The position and direction need to be consistent. Then run the command as shown in Figure 4-1-2 to open the related nodes about the bottom of the robot and the related nodes about the robot navigation.

```
wheeltec@wheeltec:~$ roslaunch turn_on_wheeltec_robot navigation.launch
```

Figure.4-1-2 roslaunch turn_on_wheeltec_robot navigation.launch

4.2 rviz navigation goal setting

The navigation target can be set programmatically, directly through rviz, or through the command line. Here, for more intuitive purposes, we directly use rviz to set the target by dragging the mouse. Run the instructions shown in Figure 4-2-1 to turn on rviz. If the target cannot be opened or dragged, it is generally caused by the incorrect IP address setting of the virtual machine. Please refer to section 4.6 for troubleshooting. Be careful to turn on rviz on the virtual machine and not on the Raspberry Pi terminal.

```

^Cpassoni@passoni:~$ rviz
[ INFO] [1591615360.192861338]: rviz version 1.13.9
[ INFO] [1591615360.192974604]: compiled against Qt version 5.9.5
[ INFO] [1591615360.193080838]: compiled against OGRE version 1.9.0 (Ghada
mon)
[ INFO] [1591615360.254701063]: Forcing OpenGL version 0.
[ INFO] [1591615360.411998239]: Stereo is NOT SUPPORTED
[ INFO] [1591615360.415702918]: OpenGL version: 3.1 (GLSL 1.4).
[ INFO] [1591615360.962317528]: Creating 1 swatches
[ INFO] [1591615361.357457823]: Creating 1 swatches

```

Figure.4-2-1 The RVIZ visualization interface opens

The displays window setup for rviz is shown in Figure 4-2-2.

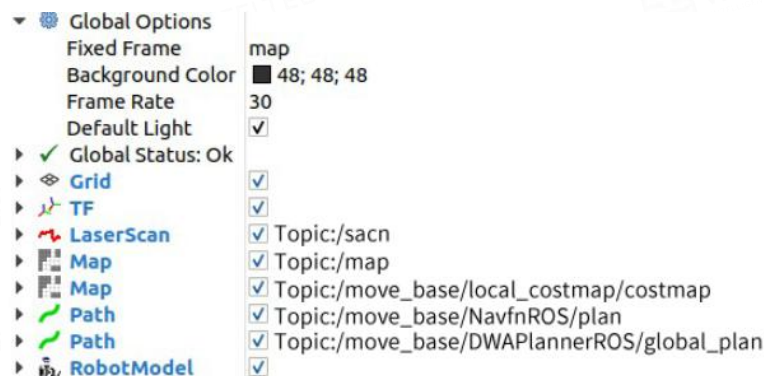


Figure. 4-2-2 RVIZ visualization interface opened



Figure.4-2-3 The TEB navigation algorithm shows path selection

Note that the topic on the right is added later and is not part of the rviz due to the display problem. In order to explain the radar and the topics that the two maps and paths listen to, we need to click on each information title and set the corresponding topic to the same as the topic on the right. Map is used to display the map; Only after

path is added can the planned path be seen. The navigation algorithm used for the two-wheel differential series is Dwa, while the algorithm used for the Ackerman, McWheel and omnidirectional series is Teb (as shown in Figure 4-2-3). Attention should be paid to differentiation when choosing.

Here costmap only shows local. The size of the local can be set using the parameters described in the next section. If we need to see global costmap, we can click add to add a map in the lower left corner and listen for the `/move_base/global_costmap/costmap` topic. Next, we set the target on the map using the 2D Nav Goal tool, and the robot can move to the specified target, as shown in Figure 4-2-4.

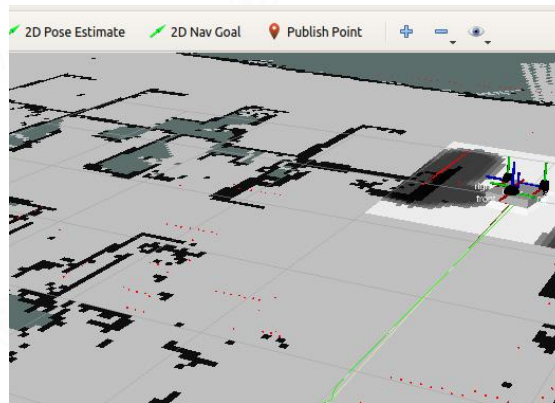


Figure.4-2-4 2D NAV Goal sets the Goal

In this map, pure black represents the obstacle, dark green surrounded by pure black represents the unexplored area, gray around pure black represents the expansion area on the obstacle, the darker the color is, the more dangerous it is, white is the safe passable area, and the track of green represents the global planning path.

4.3 Multi-point navigation

In addition to setting target navigation points using the 2D Nav Goal above, you can also set multi-point navigation using "Publish Point". The way to use it is to publish a point on the map, which is to add a navigation point, and the robot will circle between these target navigation points in order.

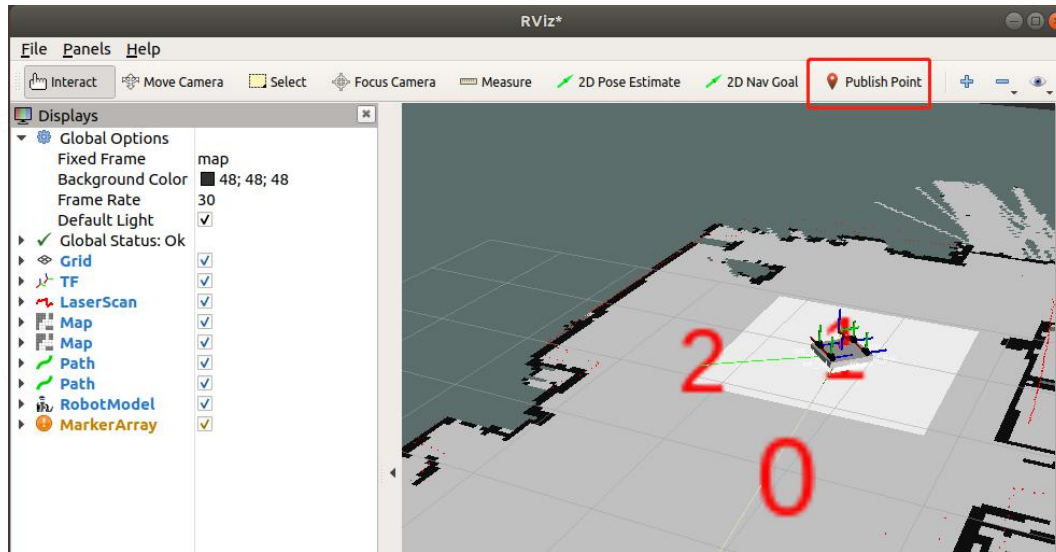


Figure.4-3-1 Publish Point sets the target

In fact, using multi-point navigation requires running a file called "sent_mark.py", which is a Python file under the "turn_on_wheeltec_robot" package. Because this Python file is already run with the navigation node, no additional run is required. If you want to end multi-point navigation, exit the navigation node by using the command "Ctrl + C" at the command line terminal.

4.4 Navigation parameter setting

In this section we explore the parameters associated with navigation packages. These files, as shown in Figure. 4-4-1, define a series of navigation-related parameters, including expansion radius, robot size, maximum and minimum robot velocity, robot acceleration, etc.

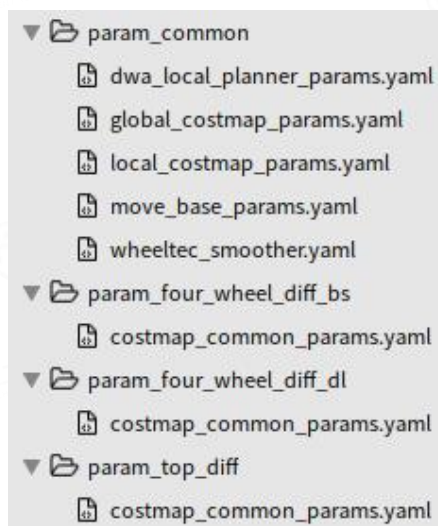


Figure.4-4-1 List of navigation parameters profile

The navigation pack uses two cost maps to store obstacle information about the map. One `global_costmap` is used for global planning, establishing long-term path planning across the environment, and the other `local_costmap` is used for local planning and obstacle avoidance. `Costmap_common_params.yaml` is the generic parameter for setting the costmap. `Dwa_local_planner_params.yaml` is responsible for sending the speed instructions calculated by the upper level planner to the lower level robot. The `costmap_common_params.yaml` file is shown in Figure 4-4-2.

```

max_obstacle_height: 2.0 # assume something like an arm is mounted on top of the robot

# Obstacle Cost Shaping (http://wiki.ros.org/costmap_2d/hydro/inflation)
# robot_radius: 0.4 # 如果机器人圆形的，注释下面的一行，开启这个
footprint: [[0.224, 0.1849], [0.224, -0.1849], [-0.224, -0.1849], [-0.224, 0.1849]] # 机器人形状
#map_type: voxel
obstacle_layer:
  enabled: true #使能障碍层
  max_obstacle_height: 2.0
  min_obstacle_height: 0.0
  #origin z: 0.0
  #z_resolution: 0.2
  #z_voxels: 2
  #unknown_threshold: 15
  #mark_threshold: 0
  combination_method: 1
  track_unknown_space: true #true needed for disabling global path planning through unknown space
  obstacle_range: 2.5 #这些参数设置了代价地图中的障碍物信息的阈值。"obstacle_range" 参数确定最大范围传感器读数
  #这会导致障碍物被放入代价地图中。在这里，我们把它设置在2.5米，这意味着机器人只会更新其地图包含距离移动基座2.5米以内的障碍物的信息。
  raytrace_range: 3.0 #"raytrace_range" 参数确定了用于清除指定范围外的空间。将其设置为3.0米。
  # 这意味着机器人将尝试清除3米外的空间，在代价地图中清除3米外的障碍物。
  #origin z: 0.0
  #z_resolution: 0.2
  #z_voxels: 2
  publish_voxel_map: false
  observation_sources: scan
  scan:
    data_type: LaserScan
    topic: "/scan"
    marking: true
    clearing: true
    expected_update_rate: 0

#cost scaling factor and inflation_radius were now moved to the inflation_layer ns
inflation_layer:
  enabled: true #使能膨胀层
  cost_scaling_factor: 5.0 # exponential rate at which the obstacle cost drops off (default: 10)
  inflation_radius: 0.3 # 机器人膨胀半径，比如设置为0.3，意味着规划的路径距离0.3米以上，这个参数理论上越大越安全
  #但是会导致无法通过狭窄的地方
static_layer:
  enabled: true
  map_topic: "/map"

```

Figure.4-4-2 costmap_common_params.yaml

The most important thing in this file is to set the shape and size of the robot, because Figure 4-3-2 describes the robot in one direction, not a circle, so we annotated `robot_radius` (line 3) and enabled footprint (line 4).

footprint: [[x0, y0], [x1, y1], ... [xn, yn]]

#robot_radius: Robot radius

It is easier to set the radius directly for a circular robot, but it is a little more complicated for a non-circular robot. The setting of footprint is shown in Figure 4-4-3. If the robot is 5-sided, it can also be expressed in a similar way.

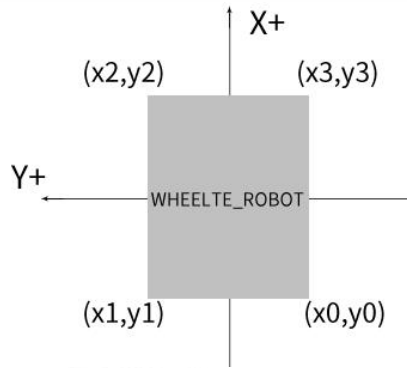


Figure.4-4-3 Footprint Settings

In addition, `inflation_radius` is the inflation radius of the robot. For example, if the `inflation_radius` is set to 0.3, it means that the planned path is more than 0.3 meters away from the robot. Theoretically, the larger the parameter is, the safer it will be, but it will make it impossible to pass through narrow places. The `dwa_local_planner_params.yaml` parameters are as follows, with important parameters shown in bold:

```

max_vel_x: 0.45 # Absolute value of maximum linear velocity in X direction, unit: m/s
min_vel_x: 0# Absolute value of the minimum linear velocity in the x direction. Negative number represents
retractable, unit: m/s
max_vel_y: 0.0 ## Absolute value of maximum linear velocity in y direction, unit: m/s. Differential driv
ing robot is 0
min_vel_y: 0.0 ## Absolute value of minimum linear velocity in y direction, in m/s. Differential driving
robot is 0
max_trans_vel: 0.5 ## Absolute value of the maximum translational velocity of the robot, in m/s
min_trans_vel: 0.1 ## Absolute value of the minimum translational velocity of the robot, which is m/s and
cannot be zero
trans_stopped_vel: 0.1# translational speed of the robot when it is considered to be in the "stopped" stat
e. If the robot's speed is lower than this value, it is considered to have stopped unit m/s
max_rot_vel: 0.7 # Absolute value of the maximum rotational angular velocity of the robot, in rad/s
min_rot_vel: 0.3 # Absolute value of the minimum rotational angular velocity of the robot, in rad/s
rot_stopped_vel: 0.4# Rotation speed of the robot when it is considered to be in the "stopped" state The u
nit is rad/s
acc_lim_x: 0.5 # Ultimate acceleration of the robot in the x direction ,in meters/SEC ^ 2
acc_lim_theta: 3.5# Limit rotational acceleration of the robot, in rad/ SEC ^2
acc_lim_y: 0.0# Ultimate acceleration of the robot in the y direction ,it is 0 for the differential robot
Goal Tolerance Parameters Target distance tolerance parameter
yaw_goal_tolerance: 0.15 # The radian tolerance (tolerance) of the controller during yaw/rotation when th
e goal point is reached. Namely, the allowable deviation angle when reaching the target point, in unit radian
s
Xy_goal_tolerance: 0.2 # Tolerance of the controller in meters in X and Y directions when it reaches the
target point. Namely, the distance error between the target point and the target point in the XY plane when i

```

t reaches the target point

#latch_xy_goal_tolerance: false # set to true means that the robot will rotate in place if it reaches the fault-tolerant distance; Even if the rotation is out of tolerance distance.

Forward Simulation Parameters Forward simulation parameters

sim_time: 1.8 # Time of forward simulation trajectory, in s(seconds)

vx_samples: 6 # The number of sampling points in the x direction velocity space

vy_samples: 1 # The number of sampling points in the y direction velocity space. Differentially driven robot always has only 1 value in Y direction (0.0)

vtheta_samples: 20 # The velocity space sampling points in the direction of rotation

Trajectory Scoring Parameters Track scoring parameter

path_distance_bias: 64.0 #The weight of the proximity of the controller to a given path

goal_distance_bias: 24.0 #The weight of the proximity of the controller to the local target point is also used for speed control

occdist_scale: 0.5 #The extent to which the controller avoids obstacles

forward_point_distance: 0.325 #Place an extra point distance with the robot as the center

stop_time_buffer: 0.2 #The minimum amount of time the robot must have before a collision occurs. The trajectory adopted during this time period is still considered valid. Namely, the length of time the robot must stop in advance to prevent a collision

scaling_speed: 0.25 #The absolute value of the speed at which the robot's footprint is scaled, in m/s.

max_scaling_factor: 0.2 #Maximum scaling factor. Max_scaling_factor is the size of the value of the above equation.

Oscillation Prevention Parameters Oscillation prevention parameter

oscillation_reset_dist: 0.05 #How many meters must the robot move to reset the oscillation mark (how far must the robot move to reset the oscillation mark)

Debugging Debug parameter

publish_traj_pc : true #The planned trajectory is visualized on RVIZ

publish_cost_grid_pc: true #Visualize the surrogate value

global_frame_id: odom_combined #Global reference coordinate system

The above comments have been very clear, you can feel the meaning of each parameter according to the effect of movement during debugging. The

global_costmap_params.yaml parameters are as follows:

global_frame: The map parameter defines the coordinate system in which the cost map should run. In this case, the /map coordinate system will be selected. For the global cost map, we use the map framework as the global framework.

robot_base_frame: The base_footprint parameter defines the coordinate system in which the cost map should be the base of the robot. This is usually either base_link or base_footprint. For our robot, base_footprint should be set.

update_frequency: 0.5 This parameter determines how often the cost map is updated (in Hz). Depending on the sensor data, the more often the global map is updated, the heavier the CPU load on your computer will be. Especially for global maps, a relatively small value between 1.0 and 5.0 is usually set.

publish_frequency: 0.5 This parameter determines the rate (in Hz) at which the cost map publishes the visual information

static_map: true # parameter determines whether the cost map is initialized by the map service provided by the map_server. If you are not using an existing map or map server, set the static_map parameter to false. This parameter is usually set to false when the local map needs to be dynamically updated based on sensor data.

The local_costmap_params.yaml parameter and the global_costmap_params.yaml parameter are mostly similar to the new ones as follows:

```
static_map: false #The parameter determines whether the map service provided by map_server is used to initialize the cost map.
```

```
rolling_window: true #Setting the parameter to true means keeping the robot in the center of the local cost map as it moves.
```

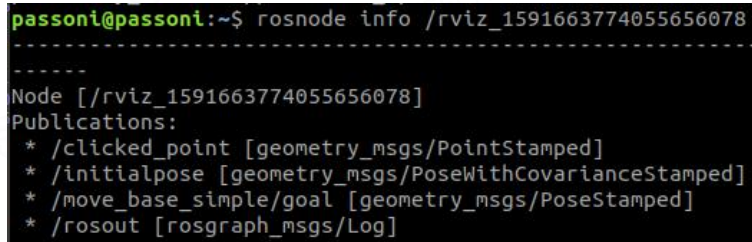
```
Width :2.0 # cost map width (m/s)
```

```
height: 2.0# Cost map height (m/s)
```

```
resolution: 0.05 # Map resolution (m/cell)
```

4.5 Navigation status monitoring and custom goals

From the previous steps, we learned how to set a goal point using rviz's 2D Nav Goal tool. Let's take a look at how this process is accomplished. Here we use the same ros command-line tool we used earlier and execute the instructions shown in Figure 4-5-1.



```
passoni@passoni:~$ rosnode info /rviz_1591663774055656078
-----
Node [/rviz_1591663774055656078]
Publications:
* /clicked_point [geometry_msgs/PointStamped]
* /initialpose [geometry_msgs/PoseWithCovarianceStamped]
* /move_base_simple/goal [geometry_msgs/PoseStamped]
* /rosout [roscpp_msgs/Log]
```

Figure.4-5-1 rosnode info /rviz_1591663774055656078

The above commands can be completed with the tab key after rviz when typed. We can see that rviz is related to a node that publishes the /move_base_simple/goal topic. Next, let's execute the instructions shown in Figure 4-5-2 to view the content of this topic. It should be noted that the content of this topic is the pose, and the topic will be updated when the new pose is set by rviz.

```
passoni@passoni:~$ rostopic echo /move_base_simple/goal
header:
  seq: 6
  stamp:
    secs: 1591665325
    nsecs: 252692812
  frame_id: "map"
pose:
  position:
    x: -0.152777194977
    y: 0.144599676132
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0795812488619
    w: 0.996828382837
```

Figure4-5-2 rostopic echo /move_base_simple/goal

You can see that the message content for this topic includes a triaxial position and a quaternion representation of the rotation state. The move_base node of the robot gets a new target pose by subscribed to the rviz publish /move_base_simple/goal topic, and then publishes the /cmd_vel topic to control the robot. Of course, we can also directly publish the topic content through rostopic pub command for navigation. Execute the instructions shown in Figure 4-5-3.

```
passoni@passoni:~$ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 1.0
    w: 0.0"
```

Figure.4-5-3 rostopic pub /move_base_simple/goal

The format of the directive to publish a topic is as follows: rostopic pub [Topic Name] [Message Type] [Parameters], When you enter the name of the topic, you can double-click the tab key to complete it automatically, and then input the target according to the actual situation, in which the frame_id needs to fill map.

4.6 Common navigation fault troubleshooting

In the process of robot navigation, some status instructions will be issued in the terminal window of launching navigation launch, as shown in Figure 4-6-1. The common status will be explained below.

```
[ INFO] [1596050306.419001801]: Got new plan
[ INFO] [1596050306.488651190]: New control command
[ INFO] [1596050306.788821801]: New control command
[ INFO] [1596050307.085795541]: Goal reached
```

Figure.4-6-1 Navigation information

New control command: It is sent by the serial port to send instructions to control the callback function of the lower machine, and it is sent once every time it enters. If you do not need this, you can go to the `wheeltec_robot.cpp` file to shield it.

Got new plan: New message on `/move_base_simple/goal` that the robot subscribed to

Goal reached: The error between the set pose and the current pose of the robot is within the allowable range set by the `dwa_local_planner_params.yaml` parameter.

DWA planner failed to produce path: The path cannot be planned according to the conditions set by the parameters, possibly because the parameters are unreasonable or there are obstacles around.

Rotate recovery behavior started: There is an obstacle near the robot, try to rotate 360° to find a new path planning.

When `rviz` sets a target through the 2D Nav Goal, it may fail to do so. For example, no green planning path appears, indicating that `/move_base_simple/Goal` has not received any new messages. This is often caused by IP incongruities, open and enter the instructions shown in Figure 4-6-2 to see the IP of the virtual machine.

```
passoni@passoni:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.114 netmask 255.255.255.0 broadcast
    inet fe80::3080:8df8:cfb0:2a6f prefixlen 64 scope
    ether 00:0c:29:45:1c:1a txqueuelen 1000 (以太网)
```

Figure.4-6-2 ifconfig

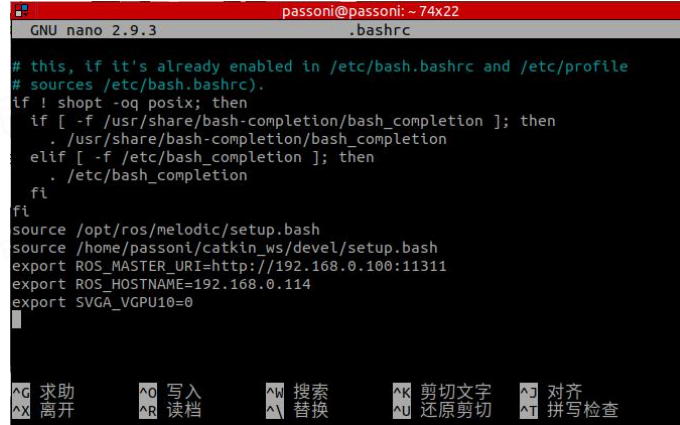
Then look at the contents of the `.bashrc` file and enter the instructions shown in Figure 4-6-3.

```
passoni@passoni:~$ nano .bashrc
```

Figure.4-6-3 nano .bashrc

Once opened, we confirm the IP of `ROS_HOSTNAME`, as shown in Figure 4-5-4. If it is consistent with what you just looked at, that's fine. If not, you need to

modify the ".bashrc" file to make it consistent. Press "Ctrl+O" to save the changes, then "Ctrl+X" to exit. After modifying the ".bashrc" file, you need to exit the terminal and reopen it or execute the command "source.bashrc" to take effect.



```
passoni@passoni: ~74x22
GNU nano 2.9.3 .bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
source /opt/ros/melodic/setup.bash
source /home/passoni/catkin_ws/devel/setup.bash
export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.114
export SVGA_VGPU10=0
^G 求助      ^O 写入      ^W 搜索      ^K 剪切文字  ^J 对齐
^X 离开      ^R 读档      ^_ 替换      ^U 还原剪切  ^T 拼写检查
```

Figure.4-6-4 .bashrc 文件