

东莞市微宏智能科技有限公司

淘宝店铺: minibalance.taobao.com

网址: www.wheeltec.net

ROS 开发教程

推荐关注我们的公众号获取更新资料



版本说明:

版本	日期	内容说明
V1.0	2020/06/10	第一次发布
V1.1	2020/07/01	第二次发布
V2.0	2020/08/01	第三次发布

序言

ROS 导航小车的全套教程包括：“运动底盘开发手册”，“Ubuntu 配置教程”，“ROS 开发手册”。关于运动底盘的 STM32 教程请查看文档“运动底盘开发手册”；关于树莓派和虚拟机开发环境搭建教程请看“Ubuntu 配置教程”。

该文档教程内容主要用于讲解 ROS 导航机器人的开发讲解，例如一些前期的准备工作以及如何建图、如何实现导航等。

在以下内容中会出现两个 Ubuntu，请注意区分：树莓派上的 Ubuntu 和虚拟机上的 Ubuntu，关于这两个系统的用户名和密码的详细信息，可以查看下表：

表 0-0

	用户名和主机名	登录密码	WiFi 名称	WiFi 密码	静态 IP
树莓派 (Jetson Nano)	wheeltec	dongguan	见表 0-1	dongguan	192.168.0.100
虚拟机	passoni	raspberry	无	无	自定义配置

表 0-1

车型	WIFI 名
差速系列	WHEELTEC88
麦轮系列	WHEELTEC66
阿克曼系列	WHEELTEC77
全向轮系列	WHEELTEC68

目录

序言.....	2
1. 固定树莓派外设串口号.....	4
2. SLAM 小车 ROS 源码解析.....	7
2.1 文件系统预览.....	7
2.2 代码构成.....	8
2.3 和下位机串口通信.....	9
2.4 ROS 话题与传感器数据发布.....	11
2.5 机器人节点解析.....	15
2.6 机器人参数解析.....	17
2.7 机器人 tf 坐标变换解析.....	18
2.8 通过 launch 文件启动机器人.....	20
3. 激光雷达建图.....	22
3.1 启动建图节点.....	22
3.2 地图的保存.....	24
4. 机器人导航.....	26
4.1 启动导航节点.....	26
4.2 rviz 导航目标设定.....	26
4.3 多点导航.....	28
4.4 导航参数设置.....	29
4.5 导航状态监控和自定义目标.....	32
4.6 导航常见故障排查.....	33

1. 固定树莓派外设串口号

在我们使用 windows 系统接一些串口外设时，通过串口调试助手或者设备管理器，我们可以看到如图 1-1、图 1-2 所示的 COM 口，这些接口一般都是变化的。

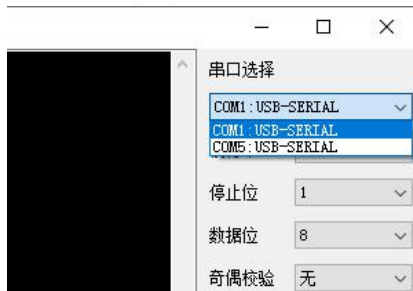


图 1-1 串口调试助手



图 1-2 设备管理器

树莓派上的程序我们一般是通过命令行执行 `roslaunch` 或者上电自动运行的，因此不太可能每次都手动选择，需要知道每个外设对应的 `ttyUSBx` (x 是 0-3)。以激光雷达为例，原本对应的是 `ttyUSB2`，重新插拔之后对应的是 `ttyUSB0`，如图 1-3 和 1-4 所示。

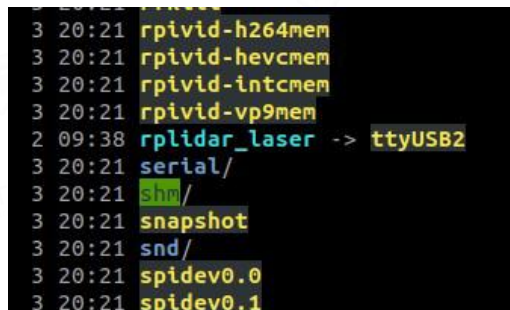


图 1-3 激光雷达 USB 口

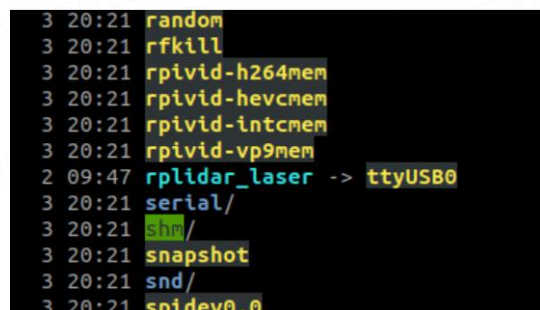


图 1-4 激光雷达重新插拔之后的 USB 口

以上信息是在树莓派终端输入命令查看，如图 1-5：



图 1-5 `ll /dev`

以上的问题如果没有解决，在调试的过程会非常麻烦。我们这里通过给 USB 设备创建别名的方式解决，输入如图 1-6 所示的指令打开以下脚本文件，如果没有就创建。

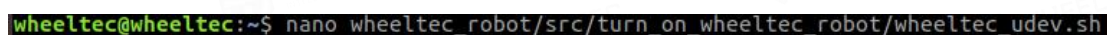
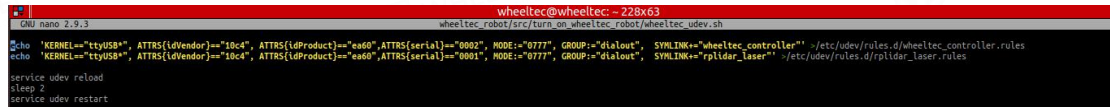


图 1-6 `nano wheeltec_robot/src/turn_on_wheeltec_robot/wheeltec_udev.sh`

打开之后如图 1-7，因为这个规则不可以换行，代码太长显得比较小，下面贴出来文字代码：



```
GNU nano 2.9.3 wheeltec@wheeltec: ~ 228x63
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
```

图 1-7 修改规则脚本

```
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
//以上两行代码在 nano 文本编辑器里面必须是一行，不可以换行。
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
//以上两行代码在 nano 文本编辑器里面必须是一行，不可以换行。
service udev reload
sleep 2
service udev restart
```

其中 idVendor 和 idProduct 是由 USB 转 TTL 芯片决定的，下面会通过软件查看。ATTRS{serial}=="0002"是串口号，这个是重点，不同的模块是不一样的，比如在这里树莓派连接 STM32 用的 CP2102 芯片（USB 线）的串口号是 2，假如更换了另外一个 CP2102 芯片（USB 线）可能是 1，所以需要通过 CP21xxCustomizationUtility 这个 windows 上的软件修改并固定，操作如图 1-8。

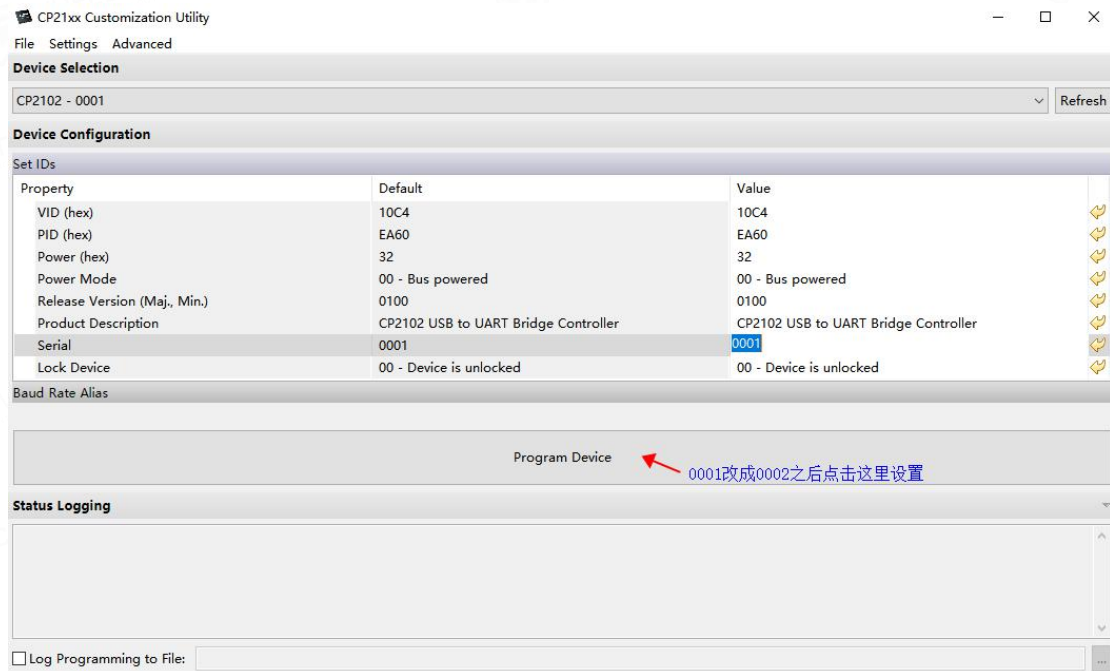


图 1-8 永久修改 CP2102 串口号

通过以上的修改之后，就可以永久固定 CP2102 的串口号，考虑到同一 USB

转 TTL 芯片的 idVendor 和 idProduct 也是固定的，这样我们通过以上的规则脚本，就可以给 USB 外设创建别名了。以上只是创建好了文件，我们需要通过图 1-9 指令赋予权限：

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

图 1-9 sudo chmod 777 wheeltec_udev.sh

然后执行该脚本文件，通过图 1-10 指令：

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

图 1-10 sudo ./wheeltec_udev.sh

之后重新插拔 USB 外设，即可实现图 1-3 和 1-4 的效果，后续不管接到那个 USB 口，使用雷达的时候我们都可以使用 rplidar_laser 代替 ttyUSBx，连接 STM32 底层的时候我们都可以使用 wheeltec_controller 代替 ttyUSBx。

2. SLAM 小车 ROS 源码解析

这章我们讲对 ROS 源码框架进行讲解。考虑到 ROS 基础教程已经比较成熟，特别是 ROS 官网非常详细，本教程侧重实例应用，更多是基于我们的机器人的源码解析。考虑到 C++ 更加高效，本套 ROS 机器人 Ubuntu 端的源码使用 C++ 做开发。

2.1 文件系统预览

通过之前的章节，我们掌握了如何使用 nfs 挂载的方式让开发更加高效便捷。在虚拟机分别输入图 2-1-1 和图 2-1-2 的指令打开树莓派主机的文件系统：

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

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

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

图 2-1-2 `subl`

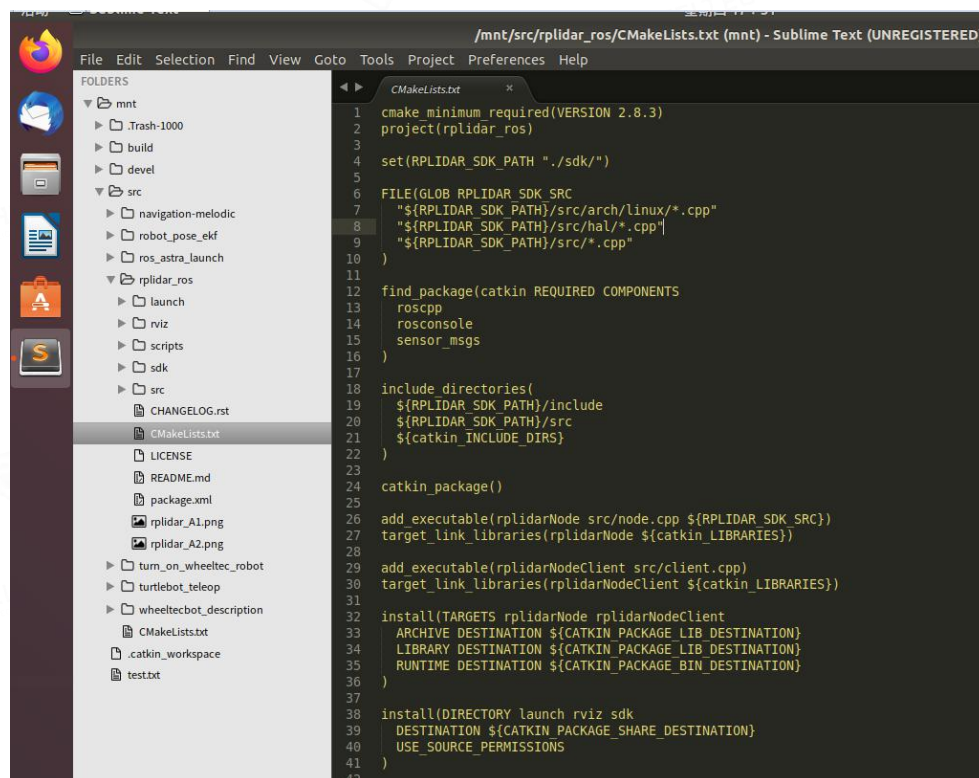


图 2-1-3 在虚拟机查看树莓派 ROS 源码

打开之后的效果如图 2-1-3。这样我们就可以像开发 STM32 一样很方便的修改了。修改之后保存，然后回到终端的“~/wheeltec_robot”目录下使用“catkin_make”这个指令编译即可。在开发过程中，如果修改了 launch 启动文

件、yaml 参数文件等可以不需要编译，保存即可重新运行；修改 cpp 文件、CMakeLists 编译规则等，需要执行上述编译命令之后再运行程序。

如果需要仅编译一个功能包，可执行如图 2-1-4 所示的指令。这样会加快编译速度，但是之后会一直只编译这个包，直到发如图 2-1-5 所示的指令。

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

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

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

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

如果还需要加快编译速度，可以开多线程，比如 catkin_make -j4 就是开 4 个线程。如果编译报 Clock skew detected 的错误，是因为 linux 系统只能编译文件修改时间比当前系统时间早的文件。需要使用 date -s 命令把树莓派的时间设置成当前的时间，因为树莓派无法联网去更新时间。命令如图 2-1-6 所示，注意使用系统当前的时间代替示意命令的时间。

```
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
```

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

2.2 代码构成

从图 2-1-3 我们可以看到，ROS 工作空间包括 build、devel、src 这 3 个文件夹和 CMakeLists 文件。

src：代码空间，主要是存放 ROS 功能包的。

build：编译空间，存放编译过程中产生的中间文件。

devel：开发空间，存放编译生成的可执行文件，比如编译出来的节点。

所以我们主要是看这个 src 文件夹里面的代码即可，src 里面是实现功能的各个元功能包和功能包，功能包数量取决于项目的复杂程度，如果只需要运行思岚 A1 激光雷达，那我们只需要加载“rplidar_ros”这个功能包，然后运行功能包里面的 launch 文件即可，这个功能包可以在思岚科技的官网下载。

同理，我们在开发其他的硬件的时候也可以在对应的官网下载功能包，ROS 相关的大部分代码都可以在 ROS 官网下载功能包。其中 ROS 源码里面的

“navigation-melodic” 导航功能包是一个元功能包 “Metapackage”，该元功能包还包括多个功能包，如图 2-2-1 和 2-2-2 所示。

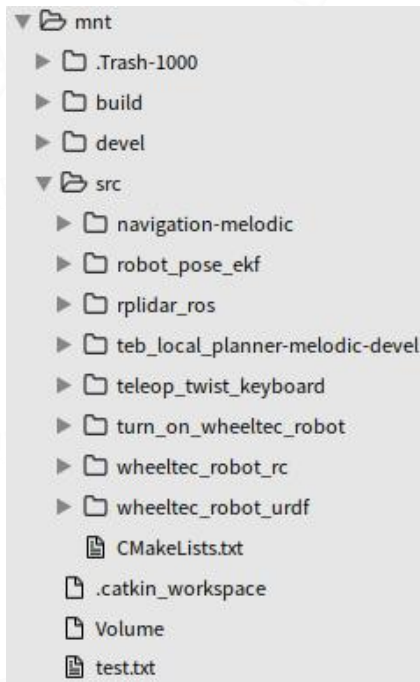


图 2-2-1 src 里面的各个功能包

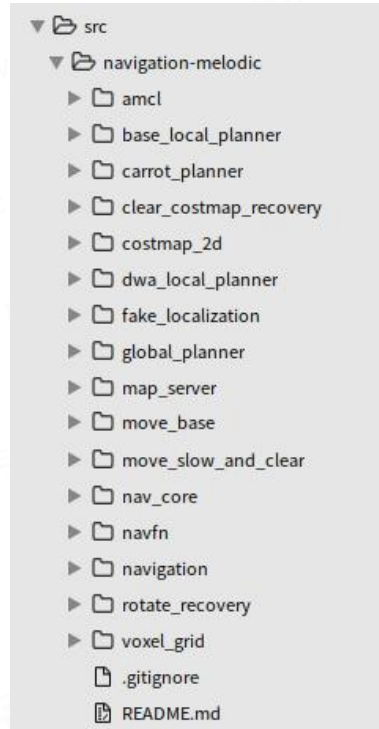


图 2-2-2 navigation-melodic 元功能包

其中和底层通信并且发布导航、建图等需要的传感器话题的功能包 “turn_on_wheeltec_robot”是我们重点需要理解的。我们注意到有.Trash-1000这个文件夹，在删除文件时，Ubuntu 将创建此类文件夹，如果您不小心删除了文件，近期的文件一般可以通过这个文件夹将其还原。

2.3和下位机串口通信

我们使用的激光雷达是通过串口和树莓派通讯的，STM32 底层下位机也是这样。其中激光雷达是通过官方功能包的“rplidarNode”节点接入 ROS 系统，这里不做更多介绍。对于开发的重点来说是，是 STM32 底层和 ROS 系统的串口通讯。

ROS 是运行在树莓派 (Jetson Nano) 上面的操作系统，机器人的建图导航等需要 IMU、里程计等数据的，理论上我们可以通过树莓派直接连接并采集这些传感器。显然，这些工作 STM32 就可以很好的完成，而且 STM32 有硬件编码器接口。我们可以使用 STM32 采集编码器和 IMU 的信息，并完成 PID 控制，然后通过串口和树莓派进行传感器数据和控制指令的交换即可。

通过前面的“STM32 运动底盘开发手册”，我们已经熟悉了 STM32 串口相关

的代码和通讯协议了。这里重点介绍 ROS 端的，代码如图 2-3-1 所示。

```
bool turn_on_robot::Get_Sensor_Data()
{
    short transition_16; //中间变量

    Stm32_Serial.read(Receive_Data.rx,sizeof(Receive_Data.rx)); //读串口数据
    Receive_Data.Frame_Header= Receive_Data.rx[0]; //数据的第一位是帧头（固定值）
    Receive_Data.Frame_Tail= Receive_Data.rx[23]; //数据的最后一位是帧尾（数据校验位）
    if (Receive_Data.Frame_Header == FRAME_HEADER )//判断帧头
    {
        if (Receive_Data.Frame_Tail == FRAME_TAIL) //判断帧尾
        {
            if (Receive_Data.rx[22] == Check_Sum(22,READ_DATA_CHECK))//校验位检测
            {
                Robot_Vel.X = Odom_Trans(Receive_Data.rx[2],Receive_Data.rx[3]); //获取底盘X方向速度
                Robot_Vel.Y = Odom_Trans(Receive_Data.rx[4],Receive_Data.rx[5]); //获取底盘Y方向速度//Y速度仅在全向移动机器人底盘有效
                Robot_Vel.Z = Odom_Trans(Receive_Data.rx[6],Receive_Data.rx[7]); //获取底盘Z方向速度
                //Robot_Vel.Z = Mpu6050.angular_velocity.z
                Mpu6050_Data.accele_x_data = IMU_Trans(Receive_Data.rx[8],Receive_Data.rx[9]); //获取IMU的X轴加速度
                Mpu6050_Data.accele_y_data = IMU_Trans(Receive_Data.rx[10],Receive_Data.rx[11]); //获取IMU的Y轴加速度
                Mpu6050_Data.accele_z_data = IMU_Trans(Receive_Data.rx[12],Receive_Data.rx[13]); //获取IMU的Z轴加速度
                Mpu6050_Data.gyros_x_data = IMU_Trans(Receive_Data.rx[14],Receive_Data.rx[15]); //获取IMU的X轴角速度
                Mpu6050_Data.gyros_y_data = IMU_Trans(Receive_Data.rx[16],Receive_Data.rx[17]); //获取IMU的Y轴角速度
                Mpu6050_Data.gyros_z_data = IMU_Trans(Receive_Data.rx[18],Receive_Data.rx[19]); //获取IMU的Z轴角速度
                //线性加速度单位转化，和STM32 MPU6050初始化的时候的量程有关
                Mpu6050.linear_acceleration.x = Mpu6050_Data.accele_x_data / ACCEL_RATIO;
                Mpu6050.linear_acceleration.y = Mpu6050_Data.accele_y_data / ACCEL_RATIO;
                Mpu6050.linear_acceleration.z = Mpu6050_Data.accele_z_data / ACCEL_RATIO;
                //陀螺仪单位转化，和STM32底层有关，这里MPU6050的陀螺仪的量程是正负500
                //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;
                //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; //（发送端将数据放大1000倍发送，这里需要将数据单位还原）
                return true;
            }
        }
    }
    return false;
}
```

图 2-3-1 ROS 系统串口接收源码

根据串口通信协议，接收数据之后先对帧头进行比较，如果相符再对帧尾进行校验，校验的思路是数据包除了帧尾的全部数据按位异或，相比于固定的帧尾，这样可以大大提高通信的可靠性，串口数据校验代码如图 2-3-2 所示。

```
unsigned char turn_on_robot::Check_Sum(unsigned char Count_Number,unsigned char mode)
{
    unsigned char check_sum=0,k;
    if(mode==0) //接收数据
    {
        for(k=1;k<Count_Number;k++)//Count_Number是接收数组位数减1
        {
            check_sum=check_sum^Receive_Data.data[k]; //按位异或
        }
    }
    if(mode==1) //发送数据
    {
        for(k=1;k<Count_Number;k++)//Count_Number是发送数组位数减1
        {
            check_sum=check_sum^Send_Data.data[k]; //按位异或
        }
    }
    return check_sum; //返回结果
}
```

图 2-3-2 串口通信校验函数

校验无误之后，我们可以依次读取数据，图 2-6 的串口接收函数其中比较特殊的是以下两行代码：

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

```
//Robot_Vel.Z = Mpu6050.angular_velocity.z;
```

Z 轴旋转角速度可由编码器间接计算得到或者陀螺仪直接采集。陀螺仪是在静止的时候会有零点漂移，编码器是高速的时候可能打滑，根据实际运行的效果评估，在对导航精度要求不太高的场合，两者的效果都可以让人满意。以上的代码是选择使用编码器的数据。

ROS 系统接收下位机发过来的数据，与此同时，ROS 系统还会向下位机发指令对机器人进行控制，主要是各轴的运动速度，如图 2-3-3。

```
void turn_on_robot::Cmd_Vel_Callback(const geometry_msgs::Twist &twist_aux)
{
    short transition; //中间变量
    Send_Data.tx[0]=FRAME_HEADER;//帧头 固定值
    Send_Data.tx[1] = 1 ; //产品型号
    Send_Data.tx[2] = 0; //机器人使能控制标志位
    //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
    //机器人y轴的目标线速度
    transition=0;
    transition = twist_aux.linear.y*1000;
    Send_Data.tx[6] = transition;
    Send_Data.tx[5] = transition>>8;
    //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);//帧尾校验位，规则参见Check_Sum函数
    Send_Data.tx[10]=FRAME_TAIL; //数据的最后一位是帧尾（固定值）
    try
    {
        Stm32_Serial.write(Send_Data.tx,sizeof (Send_Data.tx)); //向串口发数据
        ROS_INFO_STREAM("New control command");//显示受到了新的控制指令
    }
    catch (serial::IOException& e)
    {
        ROS_ERROR_STREAM("Unable to send data through serial port");//如果try失败,打印错误信息
    }
}
```

图 2-3-3 ROS 系统发指令给下位机

2.4 ROS 话题与传感器数据发布

在这节开始之前，我们先运行机器人启动的 launch 文件，命令如图 2-4-1 所示。

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

图 2-4-1 roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch

其中“roslaunch”是 ROS 命令，“turn_on_wheeltec_robot”是一个功能

包，“turn_on_wheeltec_robot.launch”是功能包下面的 launch 文件，运行之后如果没有报错，机器人就正常启动了。这个 launch 文件仅启动了机器人和底盘通讯、发布信息的相关节点，并没有启动遥控、建图、导航等节点，这样我们更容易去理解底层。

一个节点可以针对一个给定的话题发布消息，也可以关注某个话题并订阅特定类型的数据。ROS 系统提供 rostopic 命令行工具，用于显示有关 ROS 主题的调试信息，包括发布者、订阅者、发布速率和 ROS 消息。这里涉及到的消息我们不过多讲解，这里的消息可以理解成数据结构，支持标准的类型，如整形、布尔型、浮点型等，也支持嵌套结构和数组，类似于 C 语言的结构体，也可以自定义话题的消息类型。

我们先运行如图 2-4-2 所示的指令，查看系统当前的话题名称列表。注意我们这里是在虚拟机上面通过 rostopic 指令查看了树莓派主机上面的 topic，因为之前已经搭建好了多机通信的框架，包括后面的 rviz 等命令也是在虚拟机运行，这样可以缓解树莓派(Jetson Nano)主机的性能瓶颈。

```
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
```

图 2-4-2 rostopic list

机器人底层传感器数据主要是编码器里程计数据、IMU 数据、电源电压数据，其中电源电压数据不是导航必须用到的数据，虽然底层有发过来，ROS 系统接收之后可以不需要以节点的形式发出来，但是在这里我们是有发出来的，可以在机器人运行时查看这个 topic，方便在电池电压过低的时候关闭系统，如图 2-4-3。

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

图 2-4-3 rostopic echo /PowerVoltage

接下来我们运行如图 2-4-4 所示的命令，查看里程计的数据。

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

图 2-4-4 rostopic echo /odom

可以看到该 topic 的详细信息如图 2-4-5。这里我们可能会想，底层发过来的里程计就 2~3 个数据，这里怎么那么复杂呢？

```
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, 0.001, 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, 0.0, 0.0, 1000000.0, 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, 0.001, 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, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-09]
```

图 2-4-5 里程计信息

因为 ROS 作为一个开源分布式操作系统，考虑到跨平台的兼容性，里程计的数据类型是 ROS 系统固定好了的，我们只是往里面填充。其中位置信息是速度积分得到的，一般的机器人是 2 自由度或者 3 自由度，两轮差速小车是 2 自由度，全向移动小车是 3 自由度，两轮差速小车传进来的数据就是 x 方向的线速度和 z 方向的角速度，全向移动小车传进来的数据还包括 y 方向的线速度。其他的数据主要是系统当前时间、frame_id、使用的协方差等。另外，我们使用如图 2-4-6 的命令，可以查看这个 topic 发布的消息类型。这里我们用到的 rostopic echo 命令是用来查看话题的详细信息的，rostopic type 命令是查看话题的消息类型，rosmmsg show 命令是用来查看消息的数据结构。

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

图 2-4-6 rostopic type /odom

通过图 2-4-7 的命令查看“nav_msgs/Odometry”数据的结构。可以看到该数据结构和图 2-4-5 显示的结构完全一致。


```
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
```

图 2-4-7 rosmmsg show nav_msgs/Odometry

接下来我们查看 IMU 的数据，运行如图 2-4-8 所示的命令。

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

图 2-4-8 rostopic echo /mobile_base/sensors/imu_data

可以看到“IMU”这个 topic 的详细信息如图 2-4-9 所示。

```
---
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]
```

图 2-4-9 IMU 信息

这里传入的信息包括了三轴角速度和三轴线加速度，四元数是通过运算得到的。可以参考里程计的操作查看话题的消息数据类型，这里不再赘述。

2.5 机器人节点解析

Node 是 ROS 系统的节点,一个节点即为一个可执行文件,它可以通过 ROS Master 与其它节点进行通信。节点可以发布或接收一个话题,节点也可以提供或使用某种服务。rostopic 是一个命令行工具,用于显示有关 ROS 节点的调试信息,包括发布、订阅和连接。我们使用如图 2-5-1 的命令,查看系统的节点列表。

```
passoni@passoni:~$ rostopic 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
```

图 2-5-1 rostopic list

这样只能看到节点列表,不够直观,下面我们使用 rqt 工具,使用如图 2-5-2 的命令。

```
passoni@passoni:~$ rostopic run rqt_graph rqt_graph
```

图 2-5-2 rostopic run rqt_graph rqt_graph

可以看到节点之间的关系如图 2-5-3。

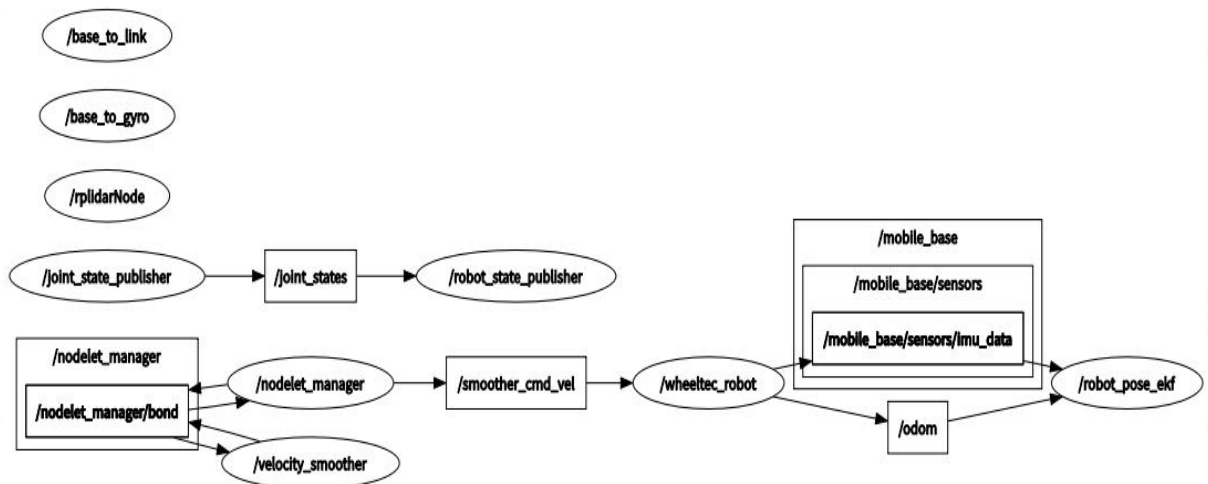


图 2-5-3 rqt 工具显示的节点列表

椭圆形的是节点名称,方框是话题,话题起到节点之间沟通的桥梁作用。ROS 的底层通信都是基于“XML-RPC”协议实现的。它允许跨平台的软件间通过发送

和接收 XML 格式的消息进行远程调用,即允许不同的操作系统、不同环境中的程序实现基于 Internet 过程调用的规范和一系列方法的实现。

目前只开启了机器人底层相关的 `turn_on_wheeltec_robot.launch` 节点,遥控、导航、建图等节点还没开启。从列表可以看到 `/rplidarNode` 等节点在不需要导航建图的时候用不到。`/joint_state_publisher` 节点是描述机器人的尺寸和轮子的位置。`/nodelet_manager` 节点是 ROS 系统提供的。

这里以 `/wheeltec_robot` 为例细讲一下节点相关的知识点。在 `turn_on_wheeltec_robot.launch` 文件里面我们可以找到节点开启的程序,其中 `turn_on_wheeltec_robot` 是节点所在的功能包, `wheeltec_robot_node` 是节点的可执行文件, `wheeltec_robot` 是运行之后的节点名称,如图 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>
```

图 2-5-4 wheeltec_robot 节点开启代码

然后我们使用如图 2-5-5 所示的命令,查看这个节点的详细信息。

```
passont@passont:~$ rostopic 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
```

图 2-5-5 rostopic info /wheeltec_robot

可以看到 `/wheeltec_robot` 这个节点有发布了 5 个话题,订阅了 2 个话题,还提供了 2 个服务。其中订阅的 2 个话题在 `wheeltec_robot.cpp` 程序里面有提供了 2 个回调函数,如图 2-5-6 所示。这里的回调函数类似于单片机开发里面的外部中断服务函数,外部中断服务函数是触发之后才会进入,回调函数是接到新

的消息了才会执行。

```
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);
}
```

图 2-5-6 订阅话题的回调函数

2.6 机器人参数解析

ROS 系统里面的参数类似于单片机开发中的全局变量，由 ROS Master 管理，其通信机制较为简单，不涉及 TCP/UDP 通信。我们可以使用“rosparam”命令去查看或者设置机器人的参数，执行如图 2-6-1 所示的指令查看目前的参数列表。

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

图 2-6-1 rosparam list

以其中一个参数作为例子讲解，/wheeltec_robot/serial_baud_rate 这个参数是通过 launch 文件设置的，有一些参数也可以 yaml 文件设置。需要注意的是，这里看到的都是上报到参数服务器的参数，在 cpp 文件里面的没有上报的局部变量是没有显示的。执行如图 2-6-2 所示的指令，可以查看该参数的值。

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

图 2-6-2 rosparam get /wheeltec_robot/serial_baud_rate

这个和我们在 launch 文件里面设置的是参数是完全一致的。如果需要修改，可以执行如图 2-6-3 所示的命令。

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

图 2-6-3 rosparam set /wheeltec_robot/serial_baud_rate 9600

可以看到，我们设置之后再查看，参数已经改变了。但是这个修改只能在 ROS 运行期间有效，当我们关闭 ROS 系统再重新打开之后，波特率又变回 115200

了。所以，如果我们需要永久修改某一个参数，需要通过修改源码，也就是 launch 或者 yaml 等文件里面的参数。我们可以通过类似的指令查看或者修改其他的参数。

2.7 机器人 tf 坐标变换解析

机器人系统通常具有随时间变化的许多 3D 坐标系，例如世界坐标系，基础坐标系等。tf 随时间跟踪所有这些框架，是处理机器人不同坐标系的一个包，机器人不同部位和世界的坐标系以 tree structure 的形式存储起来，tf 可以使任何两个坐标系之间的点、向量相互转化。

对于从单片机控制转到 ROS 上的读者，ROS 自带的 tf 可以大大加深其对机器人的理解。为了方便理解 tf 坐标变换，我们需要借助 rqt 工具，下面我们执行如图 2-7-1 的指令，可以查看机器人的 tf 树。

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

图 2-7-1 rosrn rqt_tf_tree rqt_tf_tree

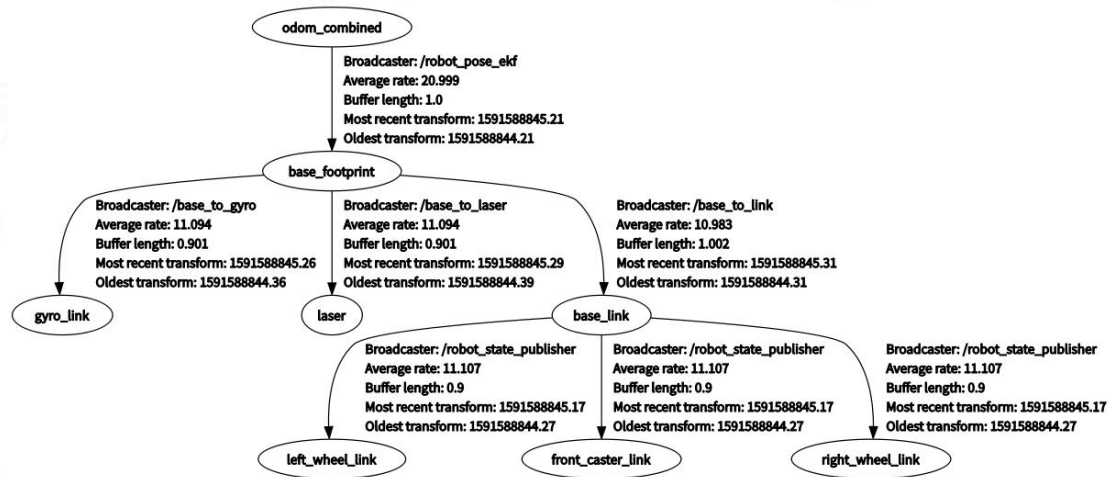


图 2-7-2 机器人 tf 树

打开之后，我们可以看到如图 2-7-2 的坐标变换关系。robot_pose_ekf 是 ROS 官方提供功能包，odom_combined 是 IMU、里程计、视觉传感器 3 者融合得到的里程计坐标系，这里的视觉传感器可以不用。base_link 与机器人中心重合，坐标系原点可以是机器人的旋转中心，base_footprint 坐标系原点为 base_link 原点在地面的投影，两者有一点区别，z 值有时候不一样。可以看到 odom_combined->base_footprint 的 tf 变换是由 robot_pose_ekf 广播的，也就

是说里程计位置的偏差是通过 ekf（拓展卡尔曼滤波）得到的。

关于 gyro_link、laser、base_link 这 3 个坐标系到 base_footprint 之间的 tf 变换，我们可以看一下如图 2-7-3 的 launch 文件里面的静态坐标变换。

```
<!-- 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>
```

图 2-7-3 机器人静态坐标变换

从第一和第二个 node 我们可以看到 gyro_link、base_link 和 base_footprint 之间是重合的，从第三个 node 我们可以看得入口参数非零，激光雷达安装在车的前部和 base_footprint 不重合。args 分别代表了 x、y、z、yaw、pitch、roll，单位分别是米和弧度。100 代表 100ms 发一次，图 2-7-2 也可以看到广播的平均频率。雷达安装的位置可能每个机器人都不一样的。所有我们使用一个自定义参数来选择不同的车型，为了更好的理解激光雷达的坐标，可以看一下图 2-7-4。图中的数据只是举例，具体每个小车的尺寸需要实测。

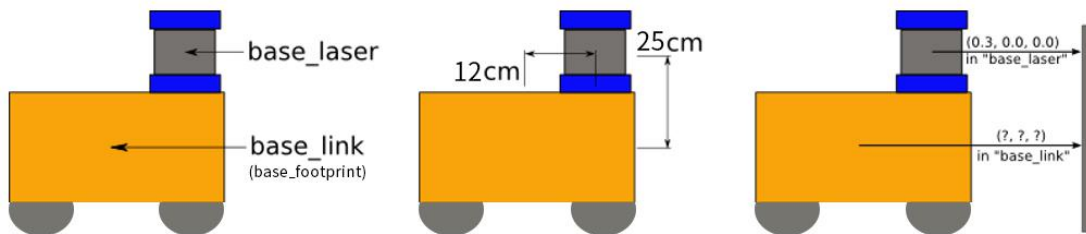


图 2-7-4 雷达和机器人坐标系的关系

Left_wheel_link、right_wheel_link、front_caster_link 这 3 个坐标系到 base_link 的坐标变换是通过 wheeltec_robot_urdf 功能包里面的 wheeltec_robot.urdf 统一机器人描述格式文件定义的。为了更直观的看到各个坐标的关系，我们在虚拟机端输入 rviz，可以看到各个坐标如图 2-7-5 所示。

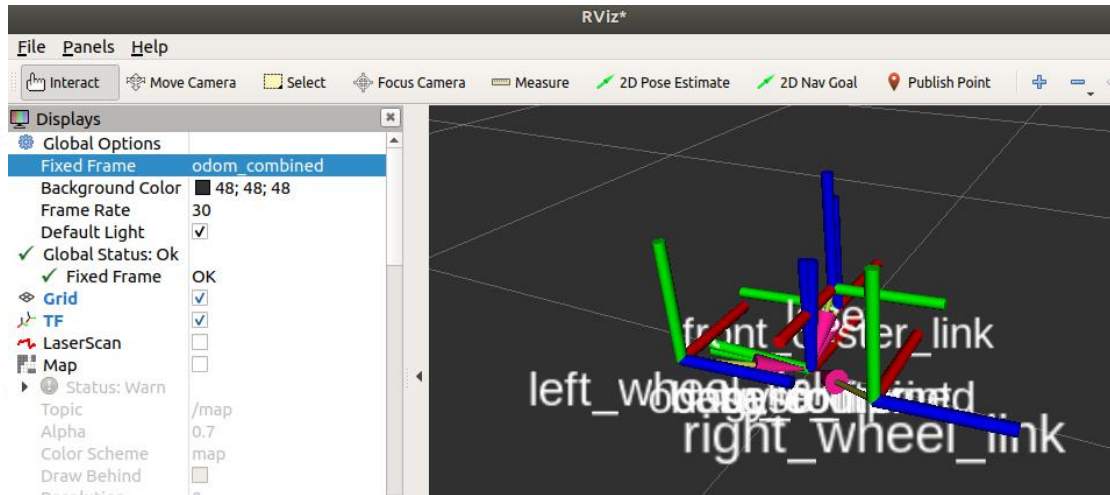


图 2-7-5 rviz 里面的坐标图示

这里因为 rviz 软件的关系，看起来比较乱，但是依然可以看到大概的坐标关系，比如 laser_link，红线是和其他相反的，说明在我们的系统里面雷达是旋转 180° 安装的。后面我们运行的时候会发现 odom_combined 会随着时间的变化而随机漂移，这是不可避免的。

2.8 通过 launch 文件启动机器人

ROS 源码写好编译无误之后，我们就可以启动了。启动 ros 节点可以使用 rosrund 命令，但是机器人一般是有多节点构成的，如果通过 rosrund 一个个地启动节点就很低效，所以我们使用 roslaunch 命令启动 launch 文件，launch 文件可以同时启动多个节点，也可以设置参数。Launch 文件列表如图 2-8-3。这里机器人启动的 launch 文件名是 turn_on_wheeltec_robot.launch。遥控机器人需要启动 wheeltec_robot_rc 功能包下面的 keyboard_teleop.launch 文件。如图 2-8-1 和 2-8-2。启动之后我们就可以通过键盘控制机器人了，遥控的时候需要确保启动这个节点的终端不是处于后台运动状态。机器人遥控节点在我们后面建图的时候可能用到。如果需要启动建图，请开启 mapping.launch 文件，如果需要导航，请开启 navigation.launch 文件。

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

图 2-8-1 roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch

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

图 2-8-2 roslaunch wheeltec_robot_rc keyboard_teleop.launch

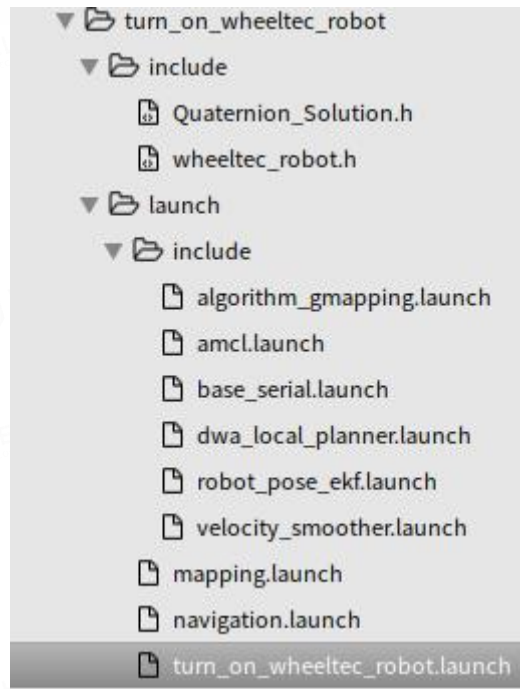


图 2-8-3 launch 文件列表

3. 激光雷达建图

ROS 开源社区汇集了多种 SLAM 算法，我们都可以直接使用或者对其进行二次开发，gmapping 是基于滤波 SLAM 框架的常用开源 SLAM 算法，也是目前最为常用和成熟的功能包。除了 gmapping 算法建图之外，我们也提供了 hector 和 karto 的建图算法。

gmapping 可以实时构建室内地图，在构建小场景地图所需的计算量较小且精度较高。相比 hector SLAM 对激光雷达频率要求低、鲁棒性高，Hector 在机器人快速转向时很容易发生错误匹配，建出的地图发生错位，原因主要是优化算法容易陷入局部最小值。而相比 cartographer 在构建小场景地图时，gmapping 不需要太多的粒子并且没有回环检测因此计算量小于 cartographer 而精度并没有差太多。gmapping 有效利用了车轮里程计信息，这也是 gmapping 对激光雷达频率要求低的原因：里程计可以提供机器人的位姿先验。karto 建图算法与 gmapping 原理相同，都是主要依赖里程计信息来完成建图。

相对于以上的两种建图算法，hector 和 cartographer 的设计初衷不是为了解决平面移动机器人定位和建图，hector 主要用于救灾等地面不平坦的情况，因此无法使用里程计。而 cartographer 是用于手持激光雷达完成 SLAM 过程，也就是说可以完全不需要里程计的信息。

3.1 启动建图节点

在运行建图节点之前我们先打开建图节点的 launch 文件查看里面的内容，可以看到支持建图的方式选择，默认是使用 gmapping 方式建图，也可以自定义修改成使用 karto 和 hector 方式建图，只需要将 default= “ ” 引号里面的内容修改后保存，然后重新运行建图节点即可。

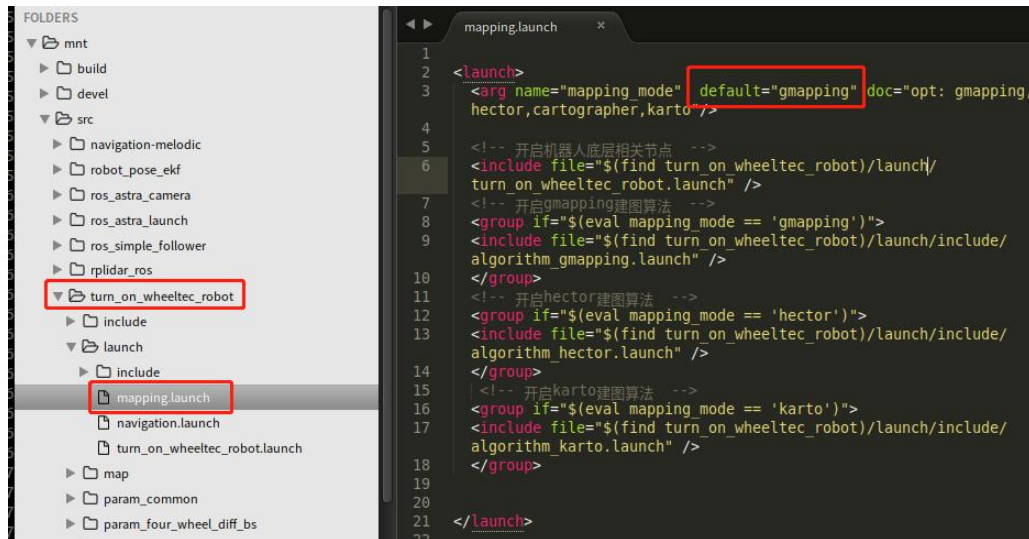


图 3-1-1 mapping.launch 文件

① gmapping

关于 gmapping 的原理，这里不做过多的讲述，无论是论文还是官网都有大量系统性的资料，下面我们直接进入应用，运行如图 3-1-2 的指令。

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

图 3-1-2 roslaunch turn_on_wheeltec_robot mapping.launch

运行之后，我们需要在虚拟机打开 rviz，通过左下角的 add 增加地图之后可以看到右边窗口有一部分建好的地图了，如图 3-1-3。

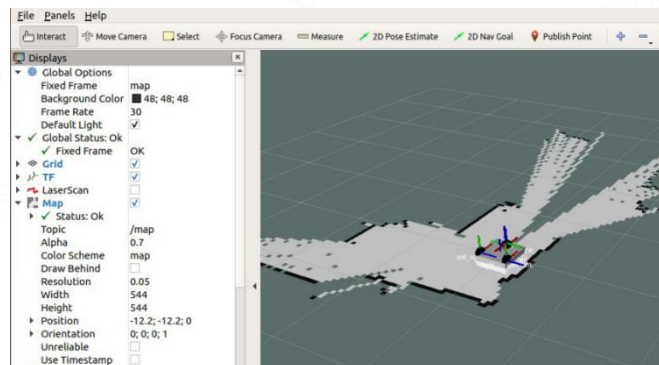


图 3-1-3 rviz 打开状态的地图

下面总结一下激光雷达建图的 2 个技巧。

技巧 1：先小闭环，后大闭环。尽量控制机器人先行走能快速闭合的小环路，然后逐步向外围扩展进行地图构建。避免直接尝试较大的环路闭环，若累计误差较大，将导致失败。避免走与当前环路闭环无关的路径，过程中会产生累计误差，容易导致环路首尾相差过大，导致无法闭环。

技巧 2：先闭环，后完善细节。避免环路闭环前，因追求建图细节进行转圈

和往复行走。在环境特征较少的情况下，此举容易导致闭环失败或错误。如需完善地图，也应先让设备直线行走快速完成闭环，随后在已闭合的路径上，进一步扫图完善细节。

一些其他的技巧可以平时自己总结或者参考一下激光雷达厂家提供的建议。这里通过 APP(键盘也行)遥控机器人，我们完成了地图的创建，如图 3-1-4。

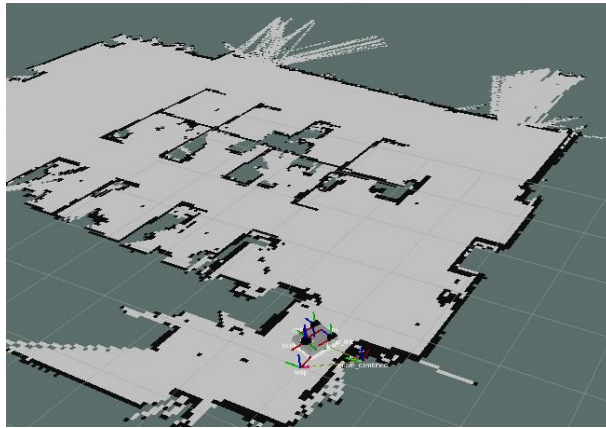


图 3-1-4 完成建图的状态

② hector 和 karto

karto 和 hector 建图的原理这里不做过多的叙述。karto 的建图原理与 gmapping 大致相同，但是使用 karto 建图时机器人的运动速度低一点建图效果要略优于 gmapping 算法。hector 建图算法完全不需要里程计的信息，因此您完全可以手持机器人完成建图，当然和另外两种方法一样直接控制机器人在空间内运动完成建图也是可以的。

3.2 地图的保存

完成上一节的建图之后先不能关闭之前的各个窗口，目前 ubuntu 的终端窗口如图 3-2-1 所示。

下面我们启动 map_server 节点对地图进行保存，进入“/home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/map”的路径执行如图 3-2-2 所示的命令。这样我们就可以把地图保存了，名称是 mymap，如果之前已有这个 mymap 地图文件，那这个指令会覆盖之前的地图。如果刚才的指令去掉 -f mymap，那么系统将以 map 作为地图名进行保存，地图会自动保存在当前终端所在的目录下。地图不仅包括一个 mymap.pgmd 地图数据文件，还不包括

一个 mymap.yaml 文件，这是一个关于地图的配置文件，其中 mymap.pgm 还可以使用 GIMP 等软件进行编辑。

```

/home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/launch/turn_on_wheeltec_robot.launch
on!
process[robot_pose_ekf-9]: started with pid [16227]
process[nodelet_manager-10]: started with pid [16235]
process[velocity_smoother-11]: started with pid [16242]
[ INFO] [1591603814.926970981]: RPLIDAR running on ROS package rplidar_ros
[ INFO] [1591603814.948432388]: wheeltec_robot serial Port opened
[ INFO] [1591603815.016148592]: output frame: odom_combined
[ INFO] [1591603815.021154962]: base frame: base_footprint
[ INFO] [1591603815.280937203]: Initializing Odom sensor
[ INFO] [1591603815.784199684]: Odom sensor activated
[ INFO] [1591603815.784523832]: Initializing Imu sensor
[ INFO] [1591603815.784641388]: Imu sensor activated
[ INFO] [1591603815.785439147]: Kalman filter initialized with odom measur
ement
RPLIDAR S/N: 92ED9A87C5E392D3A5E49EF002383065
[ INFO] [1591603817.446038368]: Firmware Ver: 1.25
[ INFO] [1591603817.447012183]: Hardware Rev: 5
[ INFO] [1591603817.448708046]: RPLidar health status : 0
[ INFO] [1591603817.998988053]: current scan mode: Express, max_distance:
12.0 m, Point number: 4.0K , angle_compensate: 1
update frame 377
update ld=0.230933 ad=0.00430127
Laser Pose= 0.314895 0.592831 -0.138302
m_count 175
Average Scan Matching Score=311.633
heff= 7.11298
Registering Scans:Done
update frame 378
update ld=0.149137 ad=0.353246
Laser Pose= 0.181492 0.659506 -0.491548
m_count 176
Average Scan Matching Score=308.465
heff= 7.11179
Registering Scans:Done
update frame 379
update ld=0.0968665 ad=0.0287538
Laser Pose= 0.0983185 0.709158 -0.520302
m_count 177
Average Scan Matching Score=309.176
heff= 7.11214
Registering Scans:Done

wheeltec@wheeltec:~$ x22
u i o
j k l
m .
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%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
currently: speed 0.2 turn 1
iiiiiiiiiiiiiiu[turtlebot_teleop_keyboard-1] process has finished cleanl
y
log file: /home/wheeltec/.ros/log/44e16a78-e95c-11ea-bb43-dca6329ddf8a/tur
tlebot_teleop_keyboard-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
wheeltec@wheeltec:~$

passoni@passoni:~$ rviz
[ INFO] [1591608464.620402776]: rviz version 1.13.9
[ INFO] [1591608464.620523406]: compiled against Qt version 5.9.5
[ INFO] [1591608464.620583297]: compiled against OGRE version 1.9.0 (Ghada
mon)
[ INFO] [1591608464.678620162]: Forcing OpenGL version 0.
[ INFO] [1591608464.814806407]: Stereo is NOT SUPPORTED
[ INFO] [1591608464.821289472]: OpenGL version: 3.1 (GLSL 1.4).
[ INFO] [1591608465.416591302]: Creating 1 swatches
[ INFO] [1591609620.122440478]: Setting goal: Frame:map, Position(0.409, 0
.808, 0.000), Orientation(0.000, 0.000, 0.000, 1.000) = Angle: 0.000

^Cpassoni@passoni:~$ rviz
[ INFO] [1591609829.619651754]: rviz version 1.13.9
[ INFO] [1591609829.619766586]: compiled against Qt version 5.9.5
[ INFO] [1591609829.619811007]: compiled against OGRE version 1.9.0 (Ghada
mon)
[ INFO] [1591609830.189835549]: Forcing OpenGL version 0.
[ INFO] [1591609830.667417505]: Stereo is NOT SUPPORTED
[ INFO] [1591609830.670787502]: OpenGL version: 3.1 (GLSL 1.4).
[ INFO] [1591609840.203363007]: Creating 1 swatches
  
```

图 3-2-1 ubuntu 终端窗口

```

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
  
```

图 3-2-2 rosrn map_server map_saver -f mymap

4. 机器人导航

导航的关键是能够实现机器人的定位和路径规划，ROS 提供了 move_base 和 amcl 这 2 个功能包作为解决方案。move_base 可以实现机器人导航中最优路径规划，amcl 可以实现二维地图中的机器人定位。我们只需要设置机器人在地图中的目标位姿，机器人即可根据地图按最优路径到达。

4.1 启动导航节点

之前的章节我们已经通过建图算法完成了 SLAM，建立了基于机器人所在环境的地图。有时候我们在 map 文件夹里面会看到很多地图，有一些地图不是在机器人所在的环境建立的，比如厂家调试的时候建立的地图是无法在客户那边使用的。所以在启动导航节点之前，我们打开 navigation.launch 文件，确认导航 launch 文件选择的地图是不是之前建立的地图 mymap，如图 4-1-1。

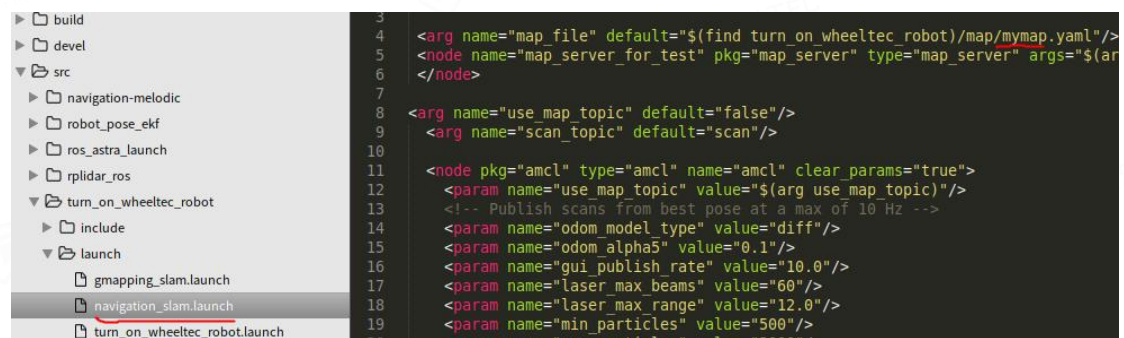


图 4-1-1 navigation_slam.launch 文件修改

修改之后我们保存，然后把之前的全部节点关闭，把机器人放在建图的起点，位置和方向都需要一致。然后运行如图 4-1-2 所示命令，开启机器人底层相关节点和机器人导航相关节点。

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

图 4-1-2 roslaunch turn_on_wheeltec_robot navigation.launch

4.2 rviz 导航目标设定

导航目标可以通过编程设置，也可以通过 rviz 直接设定，还可以通过命令行设定，这里为了更加直观，我们直接使用 rviz 通过鼠标拖拽设定目标。运行如图 4-2-1 所示指令，开启 rviz。如果遇到无法开启或者无法拖拽设置目标，一般都是虚拟机的 ip 地址设置不对导致的，可参考 4.6 节内容进行排查。注意要在虚拟

机而不是树莓派终端开启 rviz。

```
^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
```

图 4-2-1 rviz 可视化界面开启

rviz 的 displays 窗口设置如图 4-2-2。

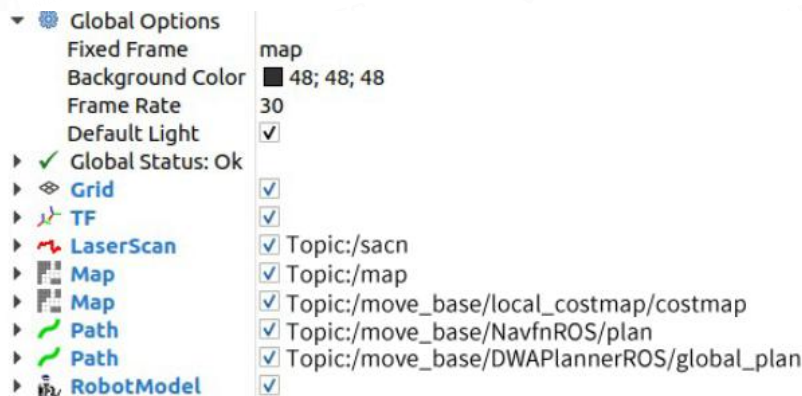


图 4-2-2 rviz 可视化界面开启



图 4-2-3 Teb 导航算法显示路径选择

注意右边的 topic 因为显示的问题，是后期加上的，不属于 rviz 的一部分，为了说明雷达和这两个 map 和 path 监听的 topic，我们需要点开每个信息标题把对应的 topic 设置的和右边一样。map 是用来显示地图的；其中 path 加上之后才可以看到规划的路径，两轮差速系列使用的导航算法是 DWA，而阿克曼、麦轮、全向轮系列使用的算法是 Teb（如图 4-2-3），在选择时要注意区分。

这里 costmap（代价地图）只显示了 local(局部)的，这个局部是多大是可以通过下一节介绍的参数设置的。如果我们需要查看 global(全局)costmap，可以在左下角点击 add 加一个 map 监听一下 /move_base/global_costmap/costmap 这个 topic。接下来我们使用 2D Nav Goal 工具在地图上设置目标，机器人即可运动到指定目标，如图 4-2-4 所示。

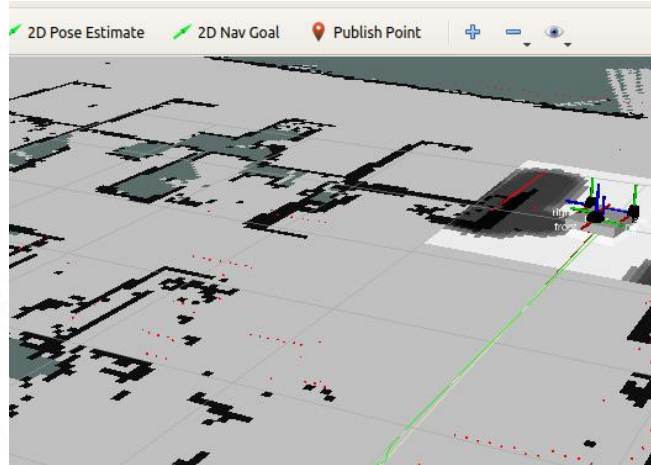


图 4-2-4 2D Nav Goal 设置目标

在这个地图中，纯黑色代表障碍物，纯黑色包围的墨绿色代表未探索区域，纯黑色周围的灰色是在障碍物上进行膨胀的区域，颜色越深代表越危险，白色是可安全通行区域，绿色的轨迹是代表全局规划路径。

4.3 多点导航

除了使用上面的 2D Nav Goal 设置目标导航点之外，还可以使用“Publish Point”设置多点导航。使用方法是每使用“Publish Point”在地图上点一个点，就是新增一个导航点，机器人会按照顺序在这些目标导航点之间循环导航。

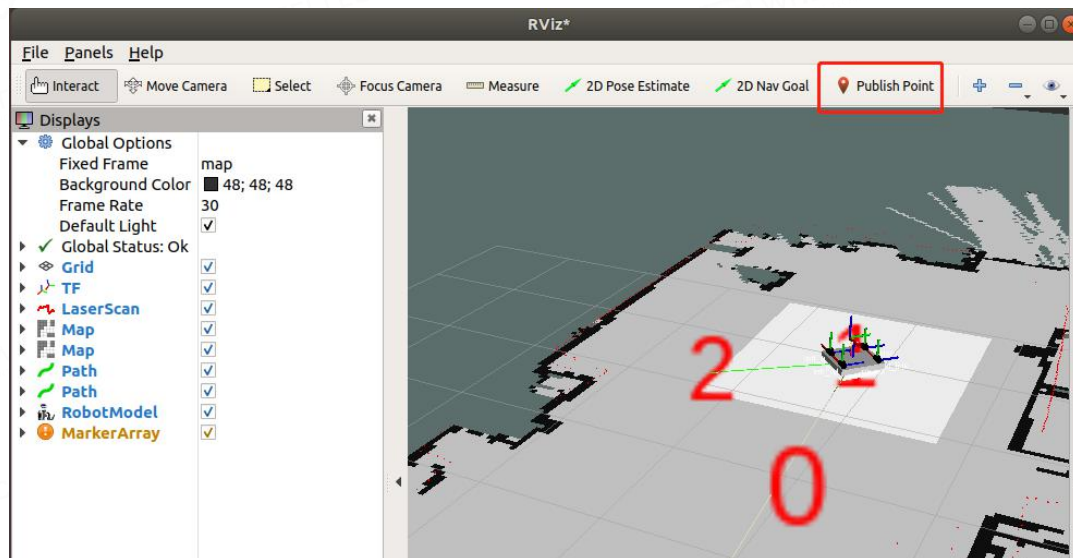


图 4-3-1 Publish Point 设置目标

实际上使用多点导航是需要运行一个“sent_mark.py”的文件，这个 python 文件在“turn_on_wheeltec_robot”功能包路径下。因为这个 Python 文件已经在运

行导航节点时一起运行了，所以不需要额外运行。如果想结束多点导航，方法是在命令行终端使用“ctrl+c”的指令退出导航节点。

4.4 导航参数设置

这节我们探索一下导航包相关的参数。如图 4-4-1 所示的这些文件定义了一系列导航相关参数，包括膨胀半径、机器人的尺寸、机器人最大最小速度、机器人加速度等。

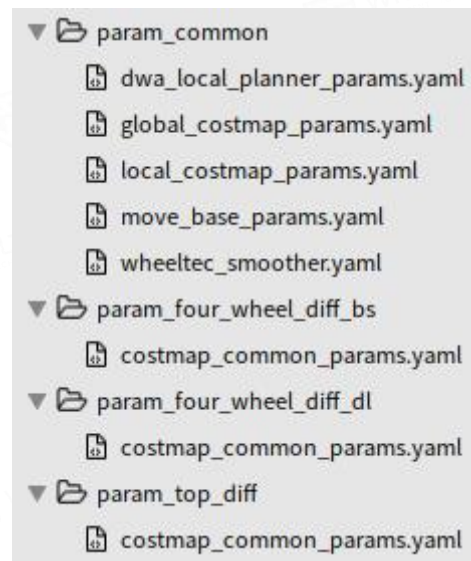


图 4-4-1 导航参数配置文件列表

导航包使用两个代价地图来存储有关地图的障碍物信息。一个 `global_costmap` 用于全局规划，在整个环境中建立远期的路径规划，另一个 `local_costmap` 用于本地规划和避障。`costmap_common_params.yaml` 是设置代价地图通用参数。`dwa_local_planner_params.yaml` 负责把上层规划器计算的速度指令发送给机器人底层。其中 `costmap_common_params.yaml` 文件如图 4-4-2 所示。

```

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
  #这将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
  #但是会导致无法闯过狭窄的地方
static_layer:
  enabled: true
  map_topic: "/map"

```

图 4-4-2 costmap_common_params.yaml

这个文件最重要的是设置机器人的形状和尺寸，因为图 4-3-2 描述的机器人是一个方向的，而不是圆形，所以我们注释了 robot_radius（第三行），开启了 footprint（第四行）。

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

#robot_radius: 机器人半径

圆形直接设置半径比较简单，非圆机器人稍微复杂一点，footprint 的设置如图 4-4-3 所示，如果机器人是 5 边形，也可以按类似的办法表示。

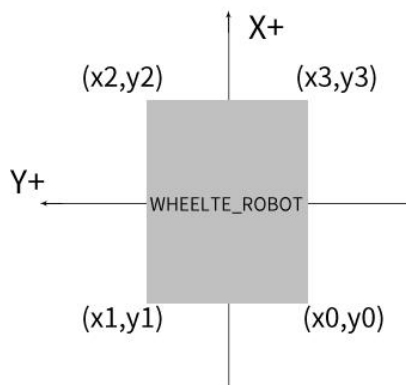


图 4-4-3 footprint 设置示意

另外 `inflation_radius` 是机器人膨胀半径, 比如设置为 0.3, 意味着规划的路径距离机器人 0.3 米以上, 这个参数理论上越大越安全, 但是会导致无法穿过狭窄的地方, 这个和机器人的结构、尺寸、地图的复杂程度都有关系。

`dwa_local_planner_params.yaml` 参数如下, 重要参数加粗显示:

```
max_vel_x: 0.45  #x 方向最大线速度绝对值, 单位:米/秒
min_vel_x: 0      #x 方向最小线速度绝对值, 负数代表可后退, 单位:米/秒
max_vel_y: 0.0    #y 方向最大线速度绝对值, 单位:米/秒。差分驱动机器人 为 0
min_vel_y: 0.0    #y 方向最小线速度绝对值, 单位:米/秒。差分驱动机器人 为 0
max_trans_vel: 0.5 # 机器人最大平移速度的绝对值, 单位为 m/s
min_trans_vel: 0.1 # 机器人最小平移速度的绝对值, 单位为 m/s 不可为零
trans_stopped_vel: 0.1 #机器人被认属于“停止”状态时平移速度。如机器人的速度低于该值, 则认为已停止单位 m/s
max_rot_vel: 0.7  #机器人的最大旋转角速度的绝对值, 单位为 rad/s
min_rot_vel: 0.3  # 机器人的最小旋转角速度的绝对值, 单位为 rad/s
rot_stopped_vel: 0.4 #机器人被认属于“停止”状态时的旋转速度。单位为 rad/s
acc_lim_x: 0.5    # 机器人在 x 方向的极限加速度, 单位为 meters/sec^2
acc_lim_theta: 3.5 #机器人的极限旋转加速度, 单位为 rad/sec^2
acc_lim_y: 0.0    #机器人在 y 方向的极限加速度, 对于差分机器人来说是 0

Goal Tolerance Parameters 目标距离公差参数

yaw_goal_tolerance: 0.15 #到达目标点时, 控制器在偏航/旋转时的弧度容差(tolerance)。即: 到达目标点时偏行角允
许的误差, 单位弧度

xy_goal_tolerance: 0.2 # 到达目标点时, 控制器在 x 和 y 方向上的容差 (tolerance) (米)。即: 到达目标点时, 在
xy 平面内与目标点的距离误差

#latch_xy_goal_tolerance: false #设置为 true 时表示: 如果到达容错距离内, 机器人就会原地旋转; 即使转动是会跑出
容错距离外。

# Forward Simulation Parameters 前向模拟参数
sim_time: 1.8      # 前向模拟轨迹的时间, 单位为 s(seconds)
vx_samples: 6      # x 方向速度空间的采样点数
vy_samples: 1      # y 方向速度空间采样点数。差分驱动机器人 y 方向永远只有 1 个值 (0.0)
vtheta_samples: 20 # 旋转方向的速度空间采样点数

# Trajectory Scoring Parameters 轨迹评分参数
path_distance_bias: 64.0      #控制器与给定路径接近程度的权重
goal_distance_bias: 24.0     #控制器与局部目标点的接近程度的权重, 也用于速度控制
occdist_scale: 0.5           # 控制器躲避障碍物的程度
forward_point_distance: 0.325 #以机器人为中心, 额外放置一个计分点的距离
stop_time_buffer: 0.2        #机器人在碰撞发生前必须拥有的最少时间量。该时间内所采用的轨迹仍视为有效。即:
为防止碰撞, 机器人必须提前停止的时间长度

scaling_speed: 0.25          #开始缩放机器人足迹时的速度的绝对值, 单位为 m/s。
max_scaling_factor: 0.2      #最大缩放因子。max_scaling_factor 为上式的值的大小。

# Oscillation Prevention Parameters 振荡预防参数
oscillation_reset_dist: 0.05 #机器人必须运动多少米远后才能复位震荡标记(机器人运动多远距离才会重置振荡标记)

# Debugging 调试参数
publish_traj_pc : true #将规划的轨迹在 RVIZ 上进行可视化
```

```
publish_cost_grid_pc: true    #将代价值进行可视化显示
```

```
global_frame_id: odom_combined #全局参考坐标系
```

以上的注释已经十分清楚，可以在调试的过程，根据运动的效果去感受每个参数的意义。`global_costmap_params.yaml` 参数如下：

`global_frame: map` 这个参数定义了代价地图应该运行的坐标系，在这种情况下，我们将选择/map 坐标系。对于全局代价地图，我们用 map 框架来作为 global 框架。

`robot_base_frame: base_footprint` 这个参数定义了代价地图应该为机器人的基座的坐标系。这个通常不是 base_link 就是 base_footprint。对于我们的机器人应设为 base_footprint。

`update_frequency: 0.5` 这个参数决定了代价地图更新的频率（以 Hz 为单位）。根据传感器数据，全局地图更新的频率，这个数值越大，你的计算机的 CPU 负担会越重，特别对于全局地图，通常设定一个相对较小、在 1.0 到 5.0 之间的值。

`publish_frequency: 0.5` 这个参数决定代价地图发布可视化信息的速率（以 Hz 为单位）。

`static_map: true` #参数确定是否由 map_server 提供的地图服务来进行代价地图的初始化。如果您没有使用现有的地图或地图服务器，请将 static_map 参数设置为 false。当本地地图需要根据传感器数据动态更新的时候，我们通常会把这个参数设为 false。

`local_costmap_params.yaml` 参数和 `global_costmap_params.yaml` 大部分类似新增部分如下：

```
static_map: false #参数确定是否由 map_server 提供的地图服务来进行代价地图的初始化。
```

```
rolling_window: true #参数设置为 true 意味着当机器人移动时，保持机器人在本地代价地图中心。
```

```
width: 2.0 #代价地图宽度（米/秒）
```

```
height: 2.0 #代价地图高度（米/秒）
```

```
resolution: 0.05 #代价地图分辨率（米/单元格）
```

4.5 导航状态监控和自定义目标

根据之前的操作，我们熟悉了如何通过 rviz 的 2D Nav Goal 工具设置目标点，具体这个过程是如何完成的呢，下面我们一探究竟。这里我们还是使用我们之前用过的 ros 命令行工具，执行如图 4-5-1 所示的指令。

```
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]
```

图 4-5-1 rosnode info /rviz_1591663774055656078

以上命令在输入时 rviz 后面可以通过 tab 键补全。我们可以看到 rviz 相关的这个节点有发布了/move_base_simple/goal 这个话题。下面我们执行如图 4-5-2 所示的指令查看这个话题的内容，需要注意的是，这个话题的内容是位姿，当 rviz 设置新的位姿的之后该话题才会更新。


```
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
```

图 4-5-2 rostopic echo /move_base_simple/goal

可以看到该话题的消息内容包含了三轴位置和四元数表示的旋转状态。机器人的 move_base 节点通过订阅 rviz 发布 /move_base_simple/goal 话题获得新的目标位姿，然后发布 /cmd_vel 话题控制机器人。当然我们也可以通过 rostopic pub 命令直接发布话题内容进行导航。执行如图 4-5-3 所示指令。

```
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"
```

图 4-5-3 rostopic pub /move_base_simple/goal[tab 键补全]

发布话题的指令格式是这样：rostopic pub [话题名称] [消息类型] [参数]，输入完话题名称的时候可以双击 tab 键自动补全，然后根据实际的情况输入目标，其中 frame_id 需要填 map。

4.6 导航常见故障排查

机器人导航的过程中，在我们开启导航 launch 的终端窗口会发布一些状态指示，如图 4-6-1 所示，下面我们对常见状态进行解释。

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

图 4-6-1 导航信息

New control command: 是串口发送指令控制下位机的回调函数发出的，每进入一次发一次，如果不需要这个可以去 wheeltec_robot.cpp 文件里面屏蔽。

Got new plan: 机器人订阅的/move_base_simple/goal 有新的消息

Goal reached: 设定的位姿和机器人当前的位姿之间的误差在 dwa_local_planner_params.yaml 参数设定的允许范围内。

DWA planner failed to produce path: 根据参数设定的条件无法规划路径, 可能是参数不合理或者周围有障碍物。

Rotate recovery behavior started: 机器人附近有障碍物, 尝试旋转 360° 找新的路径规划。

在 rivz 通过 2D Nav Goal 设置目标时, 可能出现无法成功的情况, 比如不出现绿色的规划路径, 说明/move_base_simple/goal 没有收到新的消息, 出现这个大多是 IP 不一致引起的, 打开输入如图 4-6-2 所示指令查看虚拟机的 IP。

```
passoni@passoni:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.114 netmask 255.255.255.0 broadcast
    inet6 fe80::3080:8df8:cfb0:2a6f prefixlen 64 scope
    ether 00:0c:29:45:1c:1a txqueuelen 1000 (以太网)
```

图 4-6-2 ifconfig

然后查看.bashrc 文件的内容, 输入如图 4-6-3 所示的指令。

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

图 4-6-3 nano .bashrc

打开之后我们确认一下 ROS_HOSTNAME 的 IP, 如图 4-5-4。如果是和刚才查看的一致就可以了, 如果不是, 需要修改“.bashrc”文件使其一致。修改之后按“Ctrl+O”保存, 然后按“Ctrl+X”退出。修改“.bashrc”文件之后需要退出终端重新打开或者执行“source .bashrc”这句指令才可以生效。

```
GNU nano 2.9.3      passoni@passoni: ~ 74x22
.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 VGA_VGPU10=0
```

图 4-6-4 .bashrc 文件