轮 趣 科 技

WHEELTC ROS2V3.5 使用教程

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	2021/4/19	第一次发布
V2.0	2022/06/15	更新 ROS2 版本以及功能包
V2.5	2022/11/20	更新 ROS2 传感器部分内容
V3.0	2024/04/28	更新车型切换内容
V3.5	2025/04/17	更新 slam_toolbox 简介内容

网址：www.wheeltec.net

序言

本文档主要讲解了基于 WHEELTEC 机器人的 ROS2 系统的使用方法，文档分为三个部分，包括：ROS2 概述、系统环境搭建、基于 WHEELTEC 机器人的 ROS2 功能包的使用。本文档目的是让用户能快速上手使用 ROS2 系统。

相比上一版本推出的 ROS2 WHEELTEC 机器人，本次使用的 ROS2 版本为 Galactic 版本，总体上更新了不少内容。

注意：本文档是基于 WHEELTEC 机器人搭建的 ROS2 系统，用户拿到产品时机器人端 ROS 主控已经安装好 ROS2，无需再次进行安装；同时，我们也会提供已配置好的 ROS2 虚拟机环境，若用户自行在虚拟机端配置了 ROS2 的开发环境，则请另外找我们的工作人员提供关于 RVIZ2 的功能包进行使用。

本文档教程内容默认阅读者具有 ROS1 的使用经验与基础，不建议跳过 ROS1 直接上手 ROS2。

目录

1. ROS2 概述

ROS2 的工作原理与 ROS1 类似。ROS2 是独立发行版，并不是 ROS1 的一部分，但是，它又可以使用工具嵌入 ROS1 软件包并与之协同工作。ROS1 的特征栈是用 C++编写的，客户端库是使用 C++和 Python 编写的而 ROS2 的组件是用 C 语言编写的，ROS2 中有一个用 C 语言编写的独立栈，用来连接到 ROS2 的客户端库，客户端库主要包括 rclcpp、rclpy 和 rcljava 等。

ROS2 的设计目标明确，旨在改进可用于实时系统和产品阶段解决方案的通信网络框架，其目标是：

- 支持涉及不可靠网络的多机器人系统
- 支持实时通信控制
- 跨系统平台支持
- 直接在硬件层面提供 ROS 层
- 铲除原型和最终产品之间的鸿沟

1.1 ROS2 支持的操作系统

ROS 2 支持在 Linux、Windows、macOS 以及实时操作系统（RTOS）OS 层，ROS1 只支持 Linux 和 macOS 层。ROS2 发行版及支持的操作系统如表 1-1-1 所示。注：灰色系列版本表示官方已停止维护。

表 1-1-1　ROS2 发行版及对应的操作系统

ROS2 发行版	支持的操作系统
Humble	Ubuntu 22.04; Ubuntu 20.04; macOS;Windows 10 (Visual Studio 2019);
Galactic	Ubuntu 20.04; Mac macOS 10.14 (Mojave); Windows 10 (Visual Studio 2019):
Foxy	Ubuntu20.04；macOS 10.14(sierra)；Windows10-需配置 visual studio 2019
Eloquent	Ubuntu18.04(Bionic-arm64 和 amd64)， Ubuntu18.04(Bionic-arm32)； macOS 10.12(sierra)；Windows10-需配置 visual studio 2019
Dashing	Ubuntu18.04(Bionic-arm64 和 amd64)， Ubuntu18.04(Bionic-arm32)； macOS 10.12(sierra)；Windows10-需配置 visual studio 2019
Crystal	Ubuntu18.04(Bionic)；Ubuntu 16.04(Xenial)-源码安装； macOS 10.12(sierra)；Windows10
Bouncy	Ubuntu18.04(Bionic)；Ubuntu 16.04(Xenial)-源码安装； macOS 10.12(sierra)；Windows10-需配置 visual studio 2017
Ardent	Ubuntu 16.04(Xenial)、macOS 10.12(sierra)、Windows10

1.2 ROS2 新特性

相比于 ROS1，ROS2 使用更先进的分布式架构，拥有更高的可靠性，对实时性与嵌入式设备也提供支持。二者架构如图 1-2-1 所示。

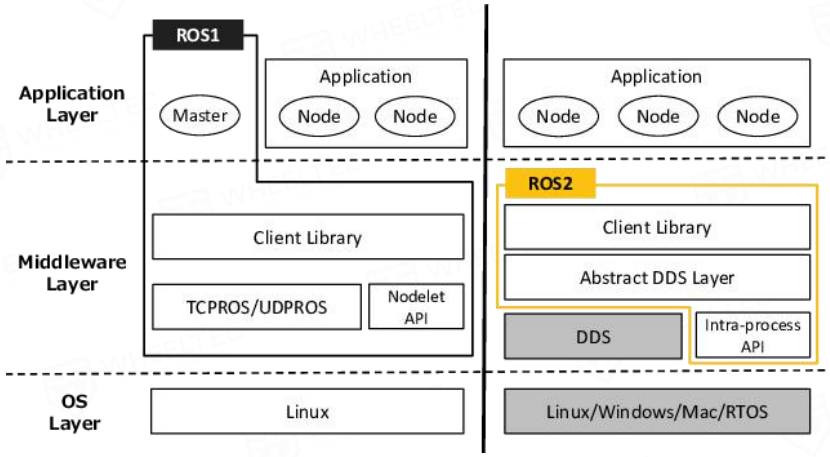

图 1-2-1　ROS1 与 ROS2 架构图

ROS 1 的通讯系统基于 TCPROS/UDPROS，强依赖于 master 节点的处理，而 ROS 2 的通讯系统是基于 DDS，进而取消了 master，同时在 ROS2 内部提供了 DDS 的抽象层与 ROS2 客户端库连接，有了这个抽象层，用户不需要去关注底层的 DDS API 的存在就可以与操作系统连接。ROS2 的 API 架构图如图 1-1-2 所示。

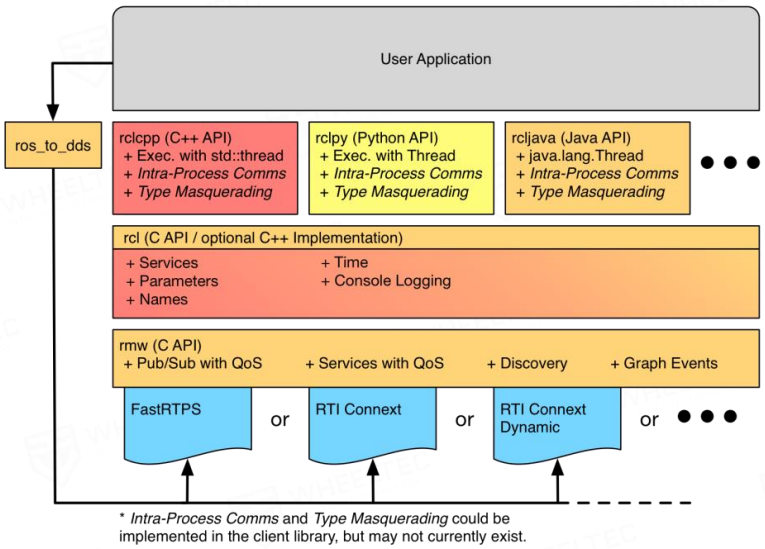

图 1-2-2　ROS2 的 API 架构图

除了以上提到的内容之外，ROS2 也推出了很多新特性，主要有：

1) 跨系统平台支持：ROS 2 支持在 Ubuntu Xenial，OS X El Capitan 以及 Windows 10 这三种平台上使用。
2) ROS2 实现了分布式架构：实现节点的分布式执行，使用 ROS 2 选择的 DDS 中间件用于数据交换，使组件可以在分布式环境中相互通信。
3) 支持实时控制。
4) 使用新版本的编程语言：ROS 2 广泛使用 C ++ 11 标准，ROS2 的 Python 版本至少为 3.5。
5) 使用了新的编译系统 Ament。
6) ROS1 可以通过 ros_bridge 和 ROS 2 进行通信。
7) 使用托管启动：用户可以指定节点启动顺序。
8) 取消了 nodelet 的概念，支持多节点初始化。
9) launch 文件使用 python 编写，相比于 xml 拓展了功能性。

2. 环境搭建

2.1 Ubuntu 版本升级

① 针对 Jetson 系列的 Ubuntu 版本升级

NVIDIA JetPack SDK 是构建 AI 应用的完整解决方案。包括 Jetson 产品的操作系统镜像、库、API、开发者工具和相关文档。在使用开发组件之前需要先进行 JetPack 安装。2022 年推出的 Jetpack5.0 版本功能对应的是 Ubuntu20.04 版本，但在项目开发过程中我们使用的还是 Jetpack4.X(Ubuntu18.04)，所以需要升级为 Ubuntu20.04 之后再安装 ROS2。

② 升级步骤

1) 烧录原版镜像 Jetpack4.6: https://developer.nvidia.com/jetpack-sdk-46

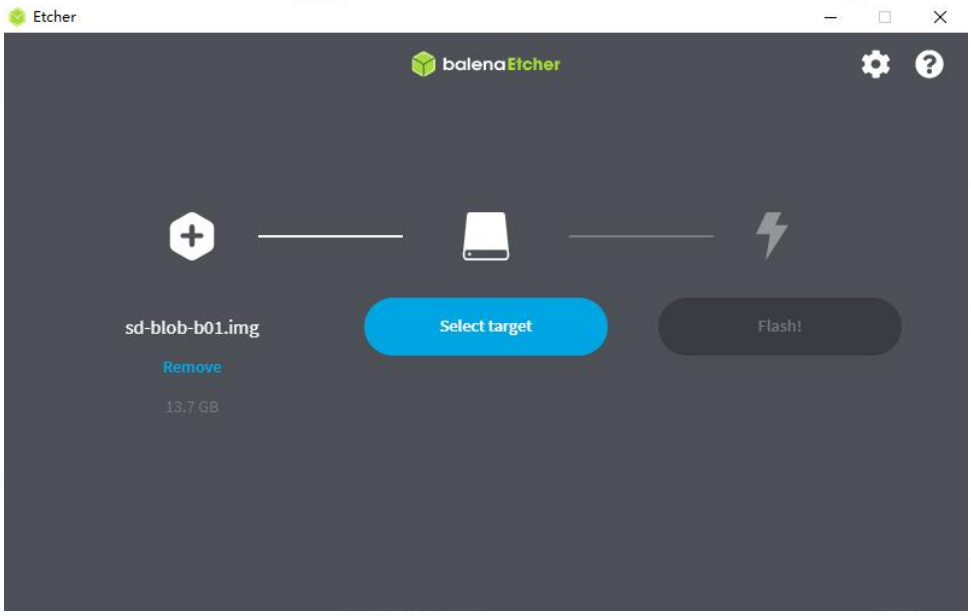

图 2-1-1 jetson nano 烧录步骤

2) 卸载不需要的软件包，例如 LibreOffice，它会使整个升级过程时间变长。选择不需要的软件后，点击 Remove 卸载。

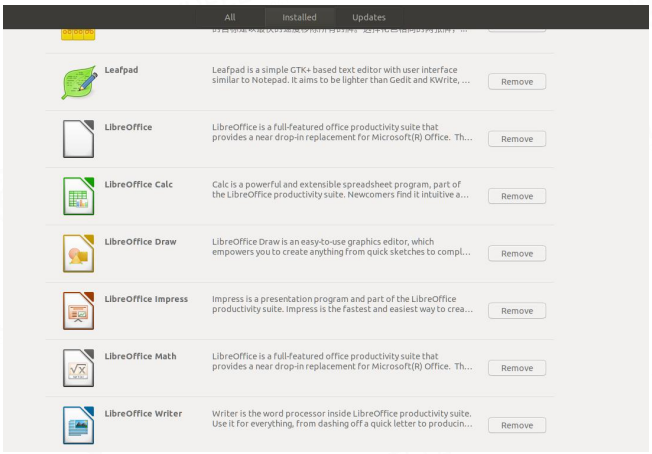

图 2-1-2　卸载系统软件页面

同时也要卸载 chromium，因为它会干扰 Snap 安装。删除不必要的软件后，需要更新、升级和清理系统：

```
#卸载 chromium
sudo apt-get purge chromium-* -y
#刷新系统
sudo apt-get update
sudo apt-get upgrade
sudo apt-get autoremove
```

重启系统：

```
sudo reboot
```

重新开机后，在/etc/update-manager/release-upgrades 文件中设置 Prompt=lts，在更新管理器中启用允许查找长期支持版更新。修改完的文件如下：

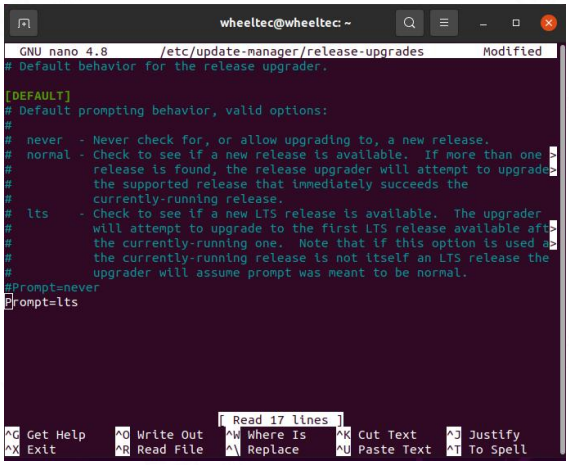

图 2-1-3　修改后的 release-upgrades 文件

设置更新管理器后，我们需要再次刷新软件数据库。完成后重新启动。

```
# 再次刷新系统
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo reboot
```

重新开机后，使用指令查看是否存在新的发行版：

```
sudo do-release-upgrade  -c
```

确认存在 20.04 之后，使用指令升级：

```
sudo do-release-upgrade
```

升级过程中需要在终端确认某些选项，使用默认值回答所有问题。

到最后一步，一些冗余软件包已被删除，Ubuntu 将要求重新启动。**在这一步输入 n, 不要重启！**

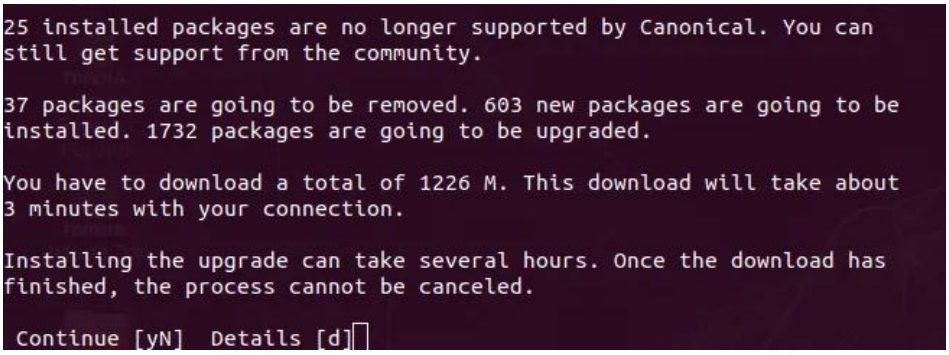

图 2-1-4　提示重启界面

重启之前要修改一些文件。

首先，在/etc/gdm3/custom.conf 文件中将 WaylandEnable=false 这一行代码注释掉；

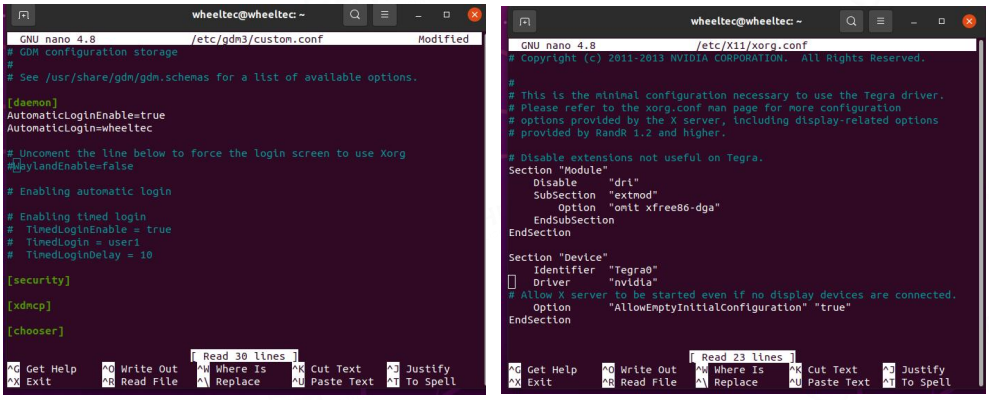

图 2-1-5　修改后的 custom.conf 文件(左)和 xorg.conf 文件(右)

然后，在/etc/X11/xorg.conf 文件中，将# Driver "nvidia"的注释去掉，

最后，在/etc/update-manager/release-upgrades 文件中将升级管理器重置为 never。

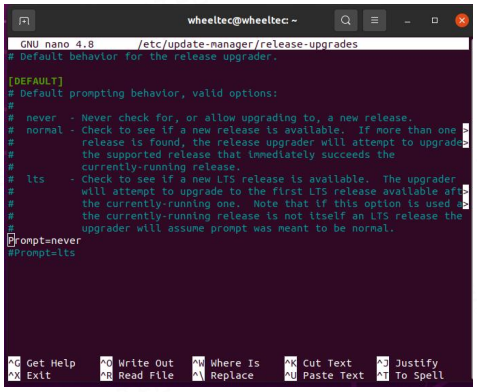

图 2-1-6　修改后的 release-upgrades 文件

重启之后即可进入 Ubuntu20.04， 删除顶部栏中扭曲的 nvidia 徽标:

```
cd /usr/share/nvpmodel_indicator
sudo mv nv_logo.svg no_logo.svg
```

注意，在 Ubuntu20.04 中，不要安装 Chromium，因为它会干扰 Snap 安装。使用预装的 Morzilla Firefox。

2.2 系统准备

① Ubuntu20.04 安装 ROS2-Galactic

有两种方法安装 ROS2，一种是源码安装，另一种是二进制安装。本教程使用的是二进制安装。

② 安装步骤

1) 设置系统区域。首先需要确保安装环境支持 UTF-8 格式，使用以下指令设置：

```
sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8 locale  # verify settings
```

执行完之后终端为：

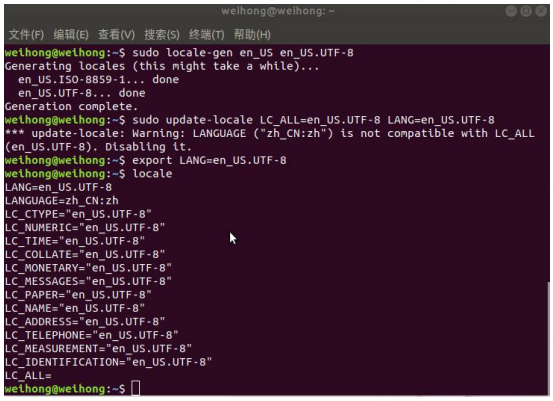

图 2-2-1　设置完系统区域的终端输出信息

设置源，将 ROS 2 apt 存储库添加到系统中，输入指令检查输出，确保启用了 Ubuntu Universe 存储库。

```
apt-cache policy | grep universe
```

应输出：

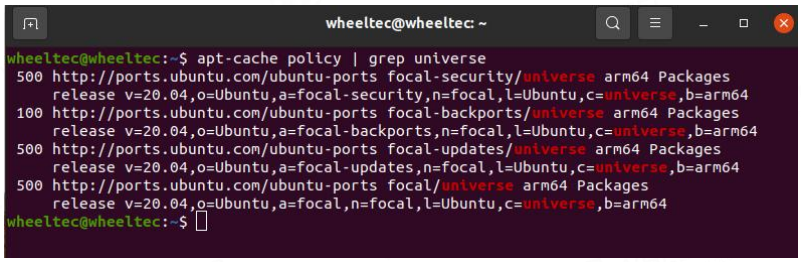

图 2-2-2　指令输出信息

```
500 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 Packages
    release v=20.04,o=Ubuntu,a=focal,n=focal,l=Ubuntu,c=universe,b=amd64
```

2）添加 ROS2 的代码仓库

使用以下指令授权密钥：

```
sudo apt update && sudo apt install curl gnupg lsb-release
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
```

将存储库添加到源列表：

```
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages.ros.org/ros2/ubuntu $(source /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

执行完后终端显示如图 2-2-3 所示。

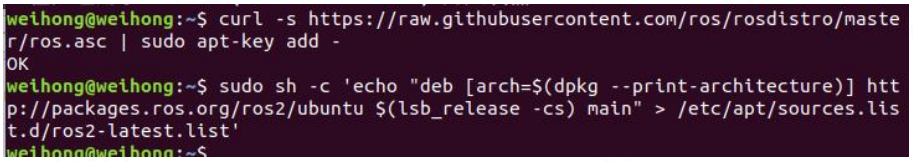

图 2-2-3　授权密钥的终端输出信息

3）更新列表：

```
sudo apt update
```

4）安装 ROS2 桌面版，包括 ROS，RViz，demos，tutorials。

```
sudo apt install ros-galactic-desktop
```

5）安装自动补全工具

```
sudo apt install -y python3-pip
pip3 install -U argcomplete
```

6）安装编译工具

```
sudo apt install python3-colcon-common-extensions
```

7）安装依赖和 ROS 工具

需要安装依赖项和工具，执行以下指令：

```
sudo apt update && sudo apt install -y \
build-essential \
cmake \
git \
python3-colcon-common-extensions \
python3-flake8 \
python3-pip \
python3-pytest-cov \
python3-rosdep \
python3-setuptools \
python3-vcstool \
wget
```

使用 pip3 安装测试功能包，执行以下指令：

```
python3 -m pip install -U \
flake8-blind-except \
flake8-builtins \
flake8-class-newline \
flake8-comprehensions \
flake8-deprecated \
flake8-docstrings \
flake8-import-order \
flake8-quotes \
pytest-repeat \
pytest-rerunfailures \
pytest \
setuptools
```

安装依赖项，执行以下指令：

```
python3 -m pip install -U importlib-metadata importlib-resources
```

③ ROS2 的环境设置

完成 ROS2 安装后，若系统安装了 ROS1，为了避免用户每次打开终端都执行一次 source 命令，可以将相关指令写进环境配置文件，使用 alias 命令设置到 bash 脚本中。两种环境的共存设置步骤为:

1) 使用 nano 编辑器打开环境配置文件：

```
nano ~/.bashrc
```

2) 在文件对应位置添加以下两句指令：

```
alias initros1=" source /opt/ros/noetic/setup.bash"
alias initros2=" source /opt/ros/galactic/setup.bash"
```

修改完后文件如图所示

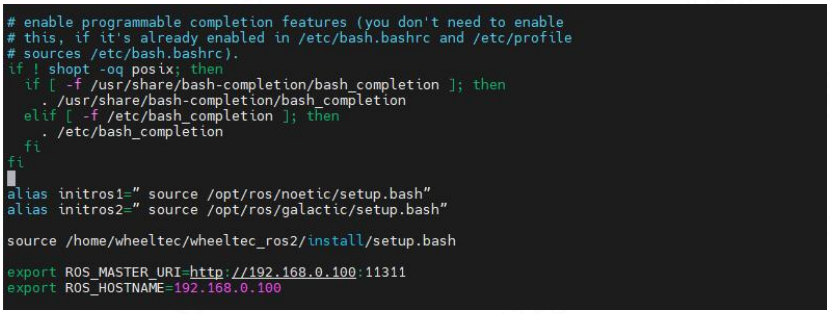

图 2-2-4 ~/.bashrc 文件内容

3) 保存退出脚本文件。使用以下命令使修改生效：

```
source ~/.bashrc
```

这个修改是为了方便用户可以根据自己的需要，在终端中输入”initros1”或”initros2”命令调用对应的系统环境。

2.3 运行测试例程

1) 使用 ctrl+alt+t 打开两个终端，输入以下命令初始化 ROS2 环境：

```
initros2
```

2) 在两个终端里分别运行系统例程的话题发布与订阅节点，命令为：

```
ros2 run demo_nodes_cpp talker
ros2 run demo_nodes_py listener
```

这里需要注意命令中的 ros2 与 run 之间的空格，执行完指令后终端输出信息如下：

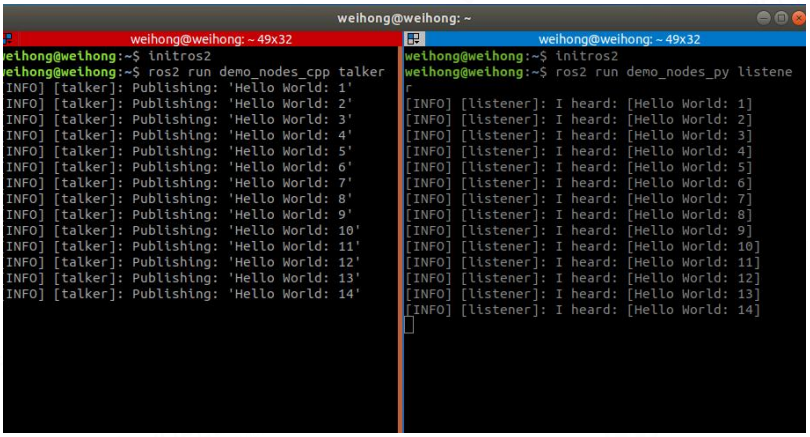

图 2-2-1　ROS2 中的发布者(左)与订阅者(右)的输出

其实 ROS1 和 ROS2 运行节点的方式没有太大区别。上述指令可以理解为：功能包的名字是 demo_nodes_cpp，节点名是 talker 和 listener，同时，这也可以验证 C ++和 Python API 是否正常工作。

3. ROS2 中的常用命令

3.1 工作空间与功能包的创建

① ROS2 工作空间的创建

工作空间是一个包含 ROS 2 功能包的目录。创建步骤为:

1) 创建一个名为 wheeltec_robot_ros2 的文件夹作为工作空间：

```
mkdir -p ~/wheeltec_robot_ros2/src
cd ~/wheeltec_robot_ros2/src
```

在 src 中可以直接拷贝放置用户的功能包，也可以在线使用 git clone 命令下载 github 中的开源代码。

② ROS2 功能包的创建

ROS 2 中使用 Ament 作为功能包的构建系统，colcon 作为它的构建工具。在 ROS2 中有 C++和 python 两种功能包，其创建指令不一样，以下教程以 C++功能包为示例。具体步骤为：

1) 进入工作空间的 src 文件夹：

```
cd ~/wheeltec_robot_ros2/src
```

2) 创建一个名为 wheeltec_ tutorials 的功能包：

```
ros2 pkg create --build-type ament_cmake wheeltec_ tutorial
```

创建语法：

```
ros2 pkg create --build-type ament_cmake <package_name>
```

此时终端输出信息如图 3-1-1 所示。

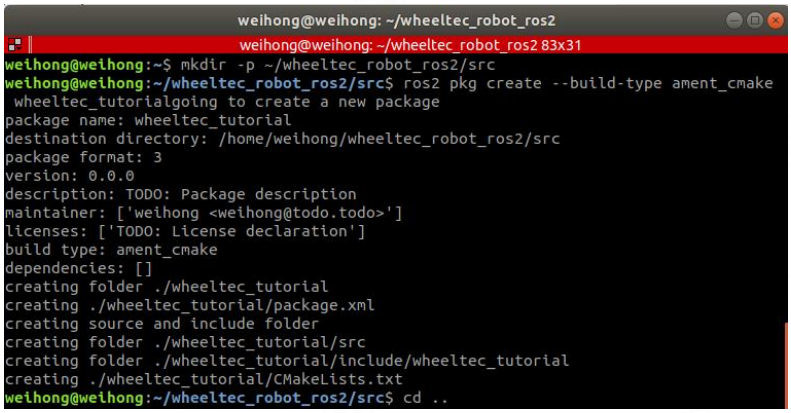

图 3-1-1 ROS2 创建一个功能包

3) 返回上一层目录后，输入以下指令进行编译。

```
colcon build#编译 src 目录中的全部功能包
colcon build --packages-select wheeltec_ tutorial#编译所选择的功能包
```

```
weihong@weihong:~/wheeltec_robot_ros2/src$ cd ..
weihong@weihong:~/wheeltec_robot_ros2$ colcon build
Starting >>> wheeltec_tutorial
Finished <<< wheeltec_tutorial [2.07s]

Summary: 1 package finished [2.19s]
weihong@weihong:~/wheeltec_robot_ros2$ colcon build --packages-select wheeltec_tuto
rial
Starting >>> wheeltec_tutorial
Finished <<< wheeltec_tutorial [0.27s]

Summary: 1 package finished [0.38s]
weihong@weihong:~/wheeltec_robot_ros2$
```

图 3-1-2　编译功能包输出示例

4) 返回到 wheeltec_robot_ros2 目录下，运行以下命令以获取工作区：

```
. install/setup.bash
```

5) 创建完成的功能包如图 3-1-2 所示。

```
weihong@weihong:~/wheeltec_robot_ros2/src$ cd wheeltec_tutorial/
weihong@weihong:~/wheeltec_robot_ros2/src/wheeltec_tutorial$ tree
.
├── CMakeLists.txt
├── include
│   └── wheeltec_tutorial
├── package.xml
└── src

3 directories, 2 files
weihong@weihong:~/wheeltec_robot_ros2/src/wheeltec_tutorial$
```

图 3-1-3　新建功能包的树结构

package.xml 文件包含有关功能包的元信息的文件，CMakeLists.txt 文件描述如何在包中生成代码。

③ ROS2 msg 文件

在 ROS2 中自定义一个消息类型，需要先创建一个新的功能包：

```
cd wheeltec_ros2/src
ros2 pkg create --build-type ament_cmake wheeltec_interfaces
```

```
wheeltec@wheeltec: ~/wheeltec_ros2/src
ROS_DISTRO was set to 'galactic' before. Please make sure that the environment
does not mix paths from different distributions.
wheeltec@wheeltec:~/wheeltec_ros2/src$ ros2 pkg create --build-type ament_cmake
 wheeltec_interfaces
going to create a new package
package name: wheeltec_interfaces
destination directory: /home/wheeltec/wheeltec_ros2/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['wheeltec <wheeltec@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./wheeltec_interfaces
creating ./wheeltec_interfaces/package.xml
creating source and include folder
creating folder ./wheeltec_interfaces/src
creating folder ./wheeltec_interfaces/include/wheeltec_interfaces
creating ./wheeltec_interfaces/CMakeLists.txt
wheeltec@wheeltec:~/wheeltec_ros2/src$
```

图 3-1-4　ROS2 自定义 msg 文件

wheeltec_interfaces是功能包的名称。目前在纯Python包中无法生成一个.msg或文件。在功能包中创建 msg 和 srv 文件：

```
mkdir msg
mkdir srv
```

进入 msg 目录，创建一个新的 Num.msg 文件，文件名首字母需大写。

```
cd msg
touch Num.msg
```

编辑 Num.msg 文件，在第一行输入

```
int64 num
```

进入 srv 目录，创建一个新的 ThreeInts.srv 文件，文件名首字母需大写。

```
cd srv
touch ThreeInts.srv
```

编辑 ThreeInts.srv 文件，输入内容：

```
int64 a
int64 b
int64 c
---
int64 sum
```

以上是自定义服务，它请求三个名为 a、b 和 c 的整数，并以一个名为 sum 的整数进行响应。

创建完文件后，此时的功能包 Tree 如下：

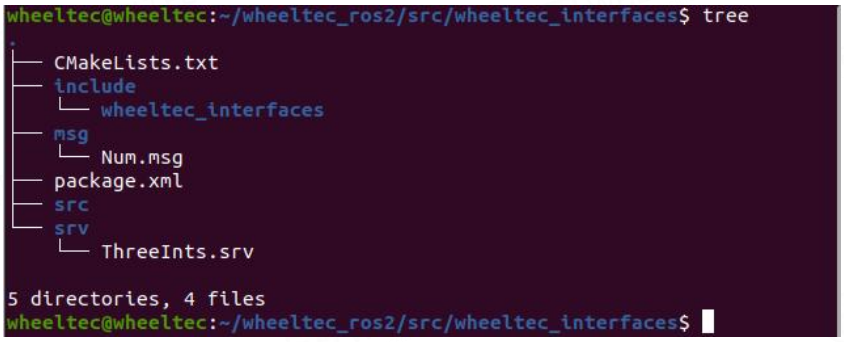

图 3-1-5　文件 Tree

修改功能包的 CMakeLists.txt 文件，添加以下内容：

```
find_package(rosidl_default_generators REQUIRED)
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "srv/ThreeInts.srv"
)
```

如图所示：

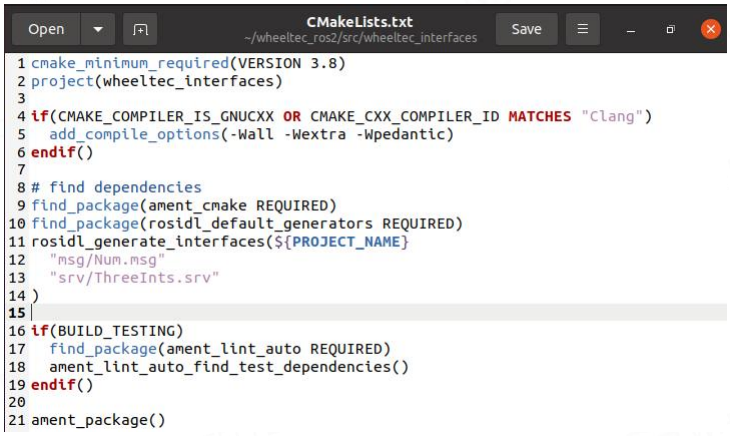

图 3-1-6　CMakeLists.txt 文件内容

修改 package.xml 文件，声明依赖。添加以下内容：

```
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

如图所示：

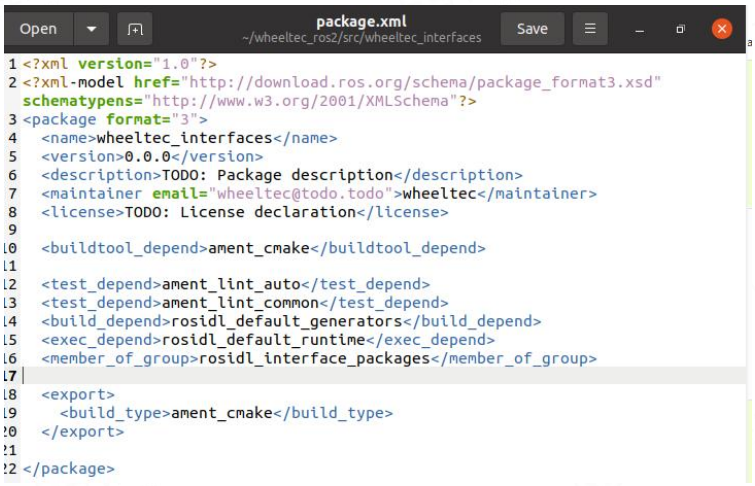

图 3-1-7　package.xml 依赖文件内容

修改完之后，回到工作空间的目录下开始编译：

```
colcon build --packages-select wheeltec_interfaces
```

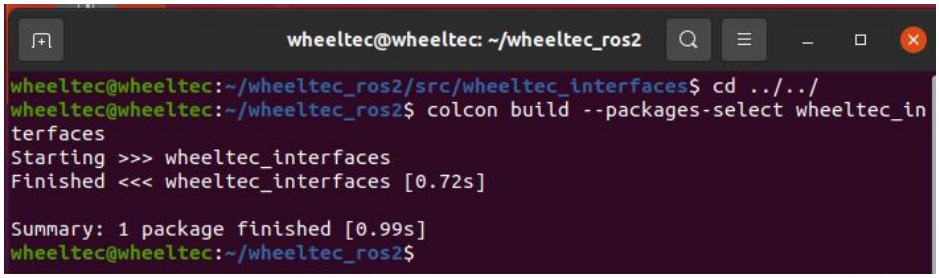

图 3-1-8　编译步骤

输入以下指令获取工作空间的可执行文件：

```
source install/setup.bash
```

在终端中，输入指令查看系统可用的 msg：

```
ros2 interface list -m
```

在终端打印的输出中可看到自定义的 msg:

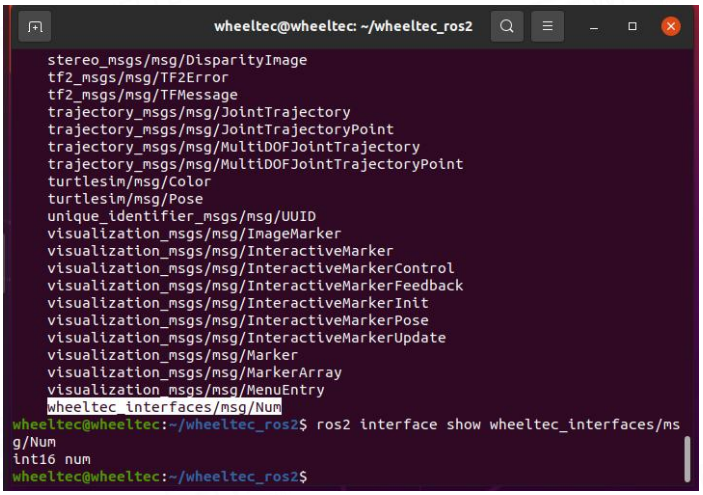

图 3-1-9　查看自定义 msg 内容

④ ROS2 确认功能包名

WHEELTEC ROS2 源码中，有些文件名并不是功能包的名字，需要自己确认功能包的具体名。以导航功能包为例，导航功能包的文件名为 wheeltec_robot_nav2，但编译功能包时，功能包名却是 wheeltec_nav2。确认功能包名方式如下：

Step1：打开 wheeltec_robot_nav2 文件，找到该目录下的 CMakeLists.txt 文件。

Step2：打开 CMakeLists.txt 文件，定位到 project 中，其中 wheeltec_nav2 则是该功能包的名字。

```
1  cmake_minimum_required(VERSION 3.5)
2  project(wheeltec_nav2)
3
```

图 3-1-10 CMakeLists.txt 文件

⑤ ROS2 功能包的编译

ROS2 中的编译构建工具变成了 colcon，工作空间的文件也与 ROS1 有所不同，ROS2 的工作空间没有 ROS1 中的 devel 文件夹；ROS2 编译工作空间下的功能包指令是：

```
colcon build
```

```
colcon build -h #查看编译子命令
colcon build --packages-select +功能包名 指定功能包编译
```

colcon 不使用源代码构建，在编译完成后工作空间中生成的文件夹如下图所示 3-1-11 所示。

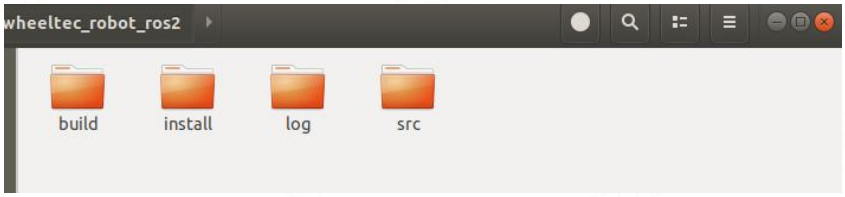

图 3-1-11　工作空间文件夹

build 文件夹：存储中间文件的位置。

install 文件夹：每个功能包的位置。

log 文件夹：所有日志信息可用的位置。

src 文件夹：源代码放置的位置。

4. ROS2 多机通信配置与虚拟机

4.1 ROS2 适配多机通信

① ROS 主控在 Ubuntu20.04 下创建热点

在 Ubuntu20.04 中调出 Network Connections 界面需要在终端输入以下指令：

```
nm-connection-editor
```

点击【+】号，选择 Wifi 选项，进行创建

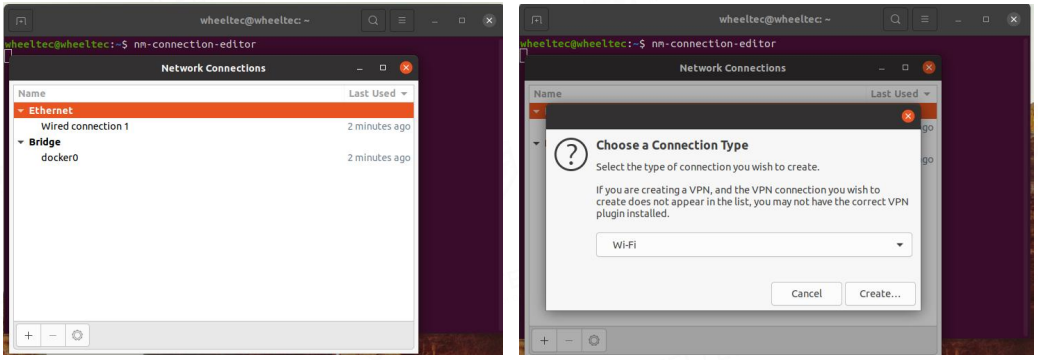

图 4-1-1　网络配置页面(左) 选择 wifi 选项(右)

在 SSID 一栏编辑热点名字 wheeltec_ros2，Mode 一栏选择 Hostpot 模式，在 WiFi Security 中选择 WPA&WPA2 Personal，设置热点密码。

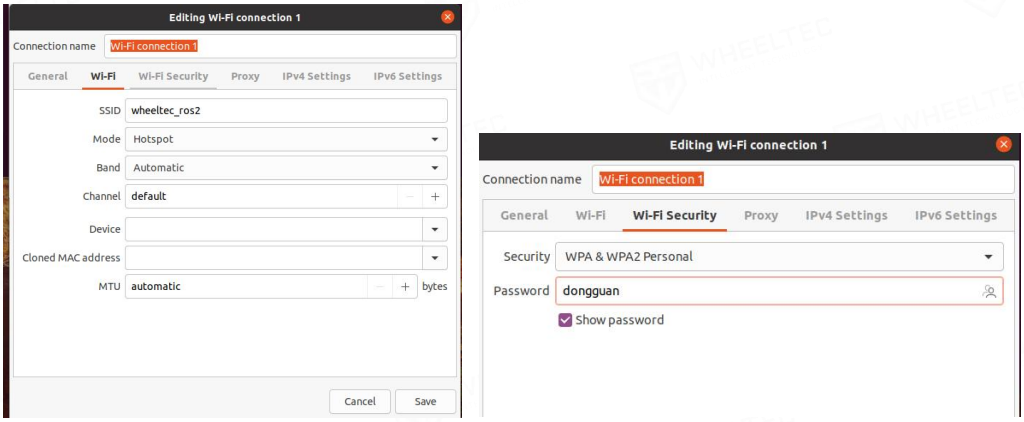

图 4-1-2　网络配置页面(左) 设置 wifi 密码(右)

在 IPV4 Setting 一栏固定 IP 地址和网关

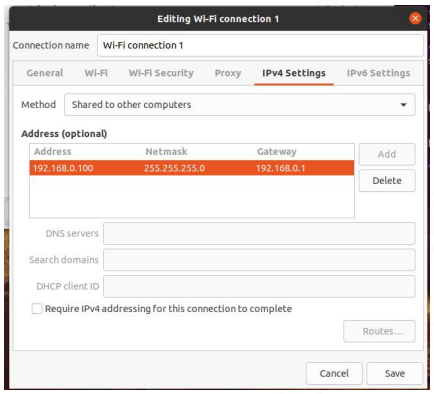

图 4-1-3　配置静态 ip 地址

完成操作后，点击右下角保存。此时在系统设置中打开 Wifi 设置，可以看到目前处于热点模式，IP 地址为 192.168.0.100，在其它设备可以搜索到这个 WiFi。

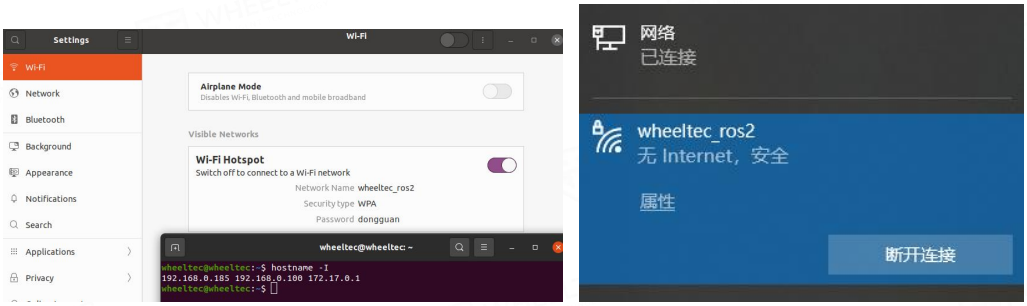

图 4-1-4　查看 ip 地址（左）连接 WHEELTEC 热点（右）

在 Windows 中连接 wheeltec_ros2 这一 WiFi，为构建多机通信。

② 虚拟机与 ROS 主控多机通信

我们提供的虚拟机适配了 Ubuntu20.04-Galactic 与 Ubuntu20.04-Noetic，虚拟机与 ROS2 WHEELTEC 搭建多机通信的步骤和 ROS1 的步骤是一致的，需要在用户目录下配置.bashrc 环境文件。

构建多机通信的前提是虚拟机/Ubuntu 电脑与 WHEELTEC 机器人在同一个局域网下。主要有两种方法：

1) 二者连接同一个 WiFi 网络实现多机通信：

虚拟机/Ubuntu 电脑连接 WHEELTEC 机器人发出的 WiFi，或者二者连接同一个 WiFi。

2) 二者连接同一个局域网下的网线实现多机通信；

在同一个局域网下，表示二者的 IP 地址网段相同。如：

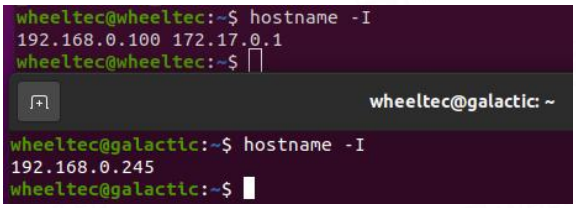

图 4-1-5　查看机器人 ip 地址（上）虚拟机 ip 地址（下）

配置.bashrc 文件：

```
nano ~/.bashrc
修改之后运行：source ~/.bashrc 生效
```

Step1:在主机端设置

WHEELTEC 机器人端默认为主机。所以将 ROS_MASTER_URI 和 ROS_HOSTNAME 的 IP 地址都设为 192.168.0.100

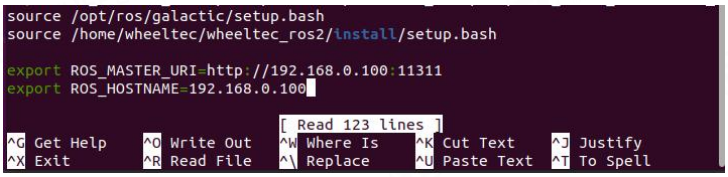

图 4-1-6　WHEELTEC 机器人~/.bashrc 文件

Setp2:在从机端配置.bashrc 文件

将 ROS_MASTER_URI 设为主机的 IP 地址，即 192.168.0.100，

ROS_HOSTNAME 的 IP 地址设为本地 IP 地址，使用 hostname –I 查看：

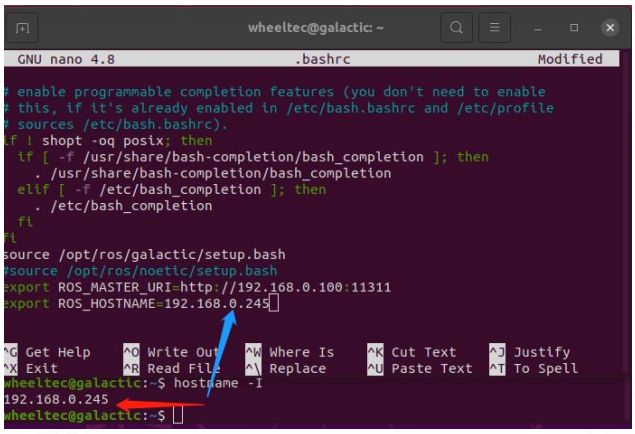

图 4-1-7　虚拟机端~/.bashrc 文件

Step3:从机端使用 SSH 远程控制主机运动，若出现以下情况，复制这一行执行之后再重新 ssh 即可。

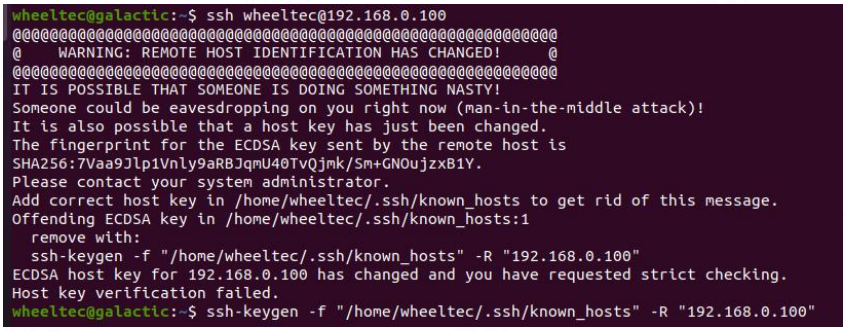

图 4-1-8　SSH 登录界面警告

Step4：验证多机通信

```
主机端运行：ros2 run demo_nodes_cpp talker
从机端运行：ros2 run demo_nodes_cpp listener
```

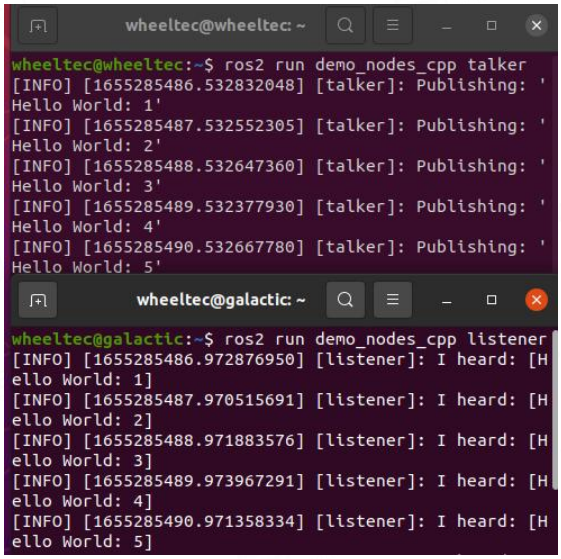

图 4-1-9　验证多机通信

4.2ROS_DOMAIN_ID 分布式通信

ROS 2 用于通信的默认中间件是 DDS。在 DDS 中，让不同逻辑网络共享物理网络的主要机制称为域 ID。同一域上的 ROS 2 节点可以自由地发现彼此并通信，而不同域上的 ROS 2 节点则不能。默认情况下，所有 ROS 2 节点使用的域 ID 为 0。只要保证在同一网络，不需要做任何配置，不同 ROS2 设备上的不同节点即可实现分布式通信。

为了避免同一网络上运行 ROS 2 的不同组计算机之间的干扰，可以为机器

人设置一个分组。

ROS2 提供了一个 DOMAIN 的机制，就类似分组一样，同一个 DOMAIN（域 ID）中的计算机才能互相通信，达到分组的目的。如果主机端和从机端分配的 ID 不同，则两者无法实现通信。

① DOMAIN 域 ID 计算规则与取值

计算规则：

- DDS 使用域 ID 来计算将用于发现和通信的 UDP 端口。网络通信时需要指定端口号，UDP 端口是一个无符号的 16 位整数。因此可以分配的范围在[0,65535]之间。
- 端口号的分配也是有规则的，根据 DDS 协议规定以 7400 作为起始端口，也即可用端口为[7400,65535]，又已知按照 DDS 协议默认情况下，每个域 ID 占用 250 个端口，那么域 ID 的个数为：(65535-7400)/250 = 232(个)，对应的其取值范围为[0,231]。
- 操作系统还会设置一些预留端口，在 DDS 中使用端口时，还需要避开这些预留端口，以免使用中产生冲突，不同的操作系统预留端口又有所差异，其最终结果是：在 Linux 下，可用的域 ID 为[0,101]与[215-231]，在 Windows 和 Mac 中可用的域 ID 为[0,166]，综上，为了兼容多平台，建议域 ID 在[0,101]范围内取值。
- 每个域 ID 默认占用 250 个端口，且每个 ROS2 节点需要占用两个端口。另外，按照 DDS 协议每个域 ID 的端口段内，第 1、2 个端口是 Discovery Multicast 端口与 User Multicast 端口，从第 11、12 个端口开始是域内第一个节点的 Discovery Unicast 端口与 User Unicast，后续节点所占用端口依次顺延，那么一个域 ID 中的最大节点个数为：(250-10)/2 = 120(个)。

取值建议：

- ROS_DOMAIN_ID 的取值在[0,101] 之间，包含 0 和 101；
- 每个域 ID 内的通信节点总数是有限制的，需要小于等于 120 个；
- 如果域 ID 为 101，那么该域的节点总数需要小于等于 54 个。

② DOMAIN 域 ID 设置

在主机（机器人）端和从机（虚拟机）端的~/.bashrc 文件中加入以下配置指令，即可将两者分配到一个小组中，若临时分配用户组可直接在终端里输入该指令：

```
export ROS_DOMAIN_ID=<your_domain_id>
```

机器人端执行：

```
echo "export ROS_DOMAIN_ID=10" >> ~/.bashrc
# 这里的 10 是 ROS_DOMAIN_ID，不一定要用 10，符合 ROS_DOMAIN_ID 的规则即可
source ~/.bashrc
ros2 run demo_nodes_py talker
```

虚拟机端执行：

```
echo "export ROS_DOMAIN_ID=10" >> ~/.bashrc
# 这里的 1 是 ROS_DOMAIN_ID,需要和主机端的值保持一致
source ~/.bashrc
ros2 run demo_nodes_py listener
```

实现通信：

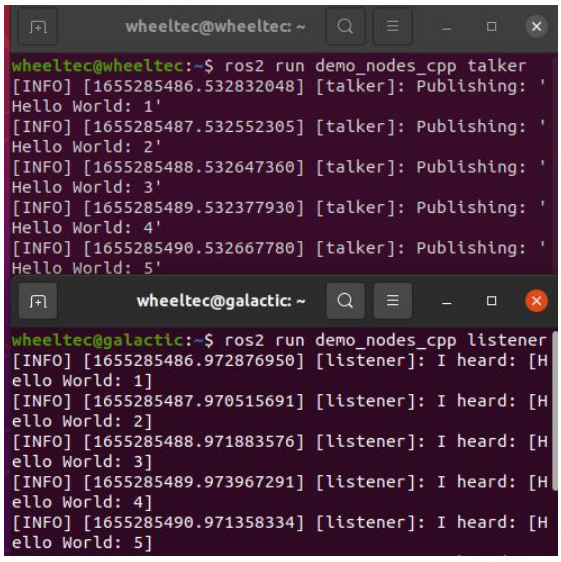

图 4-2-1　验证多机通信

4.3 虚拟机端功能包适配

在使用 WHEELTEC 机器人时，一般使用虚拟机或 Ubuntu 电脑为客户端，方便对 WHEELTEC 机器人进行远程控制。若用户使用出厂提供的 ROS2 虚拟机，则不需要进行适配，若用户自己的虚拟机安装了 ROS2 系统，则需要适配两个功能包。以下演示基于显示屏界面操作。

① RVIZ2 可视化界面

将功能包 wheeltec_rviz2 拷贝到工作空间下的 src 文件中，然后编译。

```
colcon build --packages-select wheeltec_rviz2
```

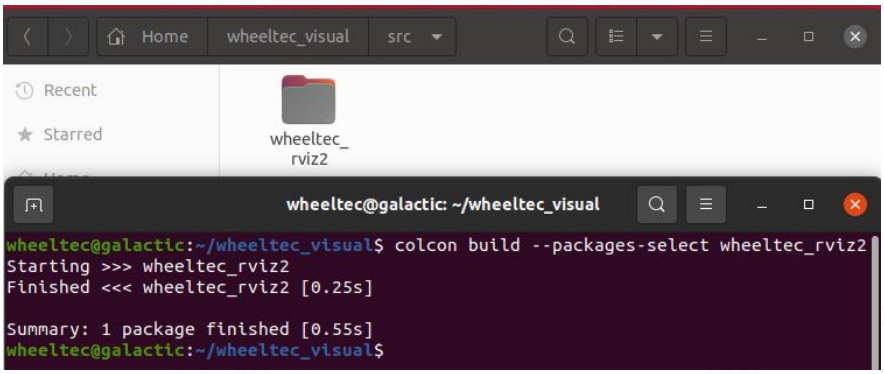

图 4-2-1　编译 wheeltec_rviz2 功能包

当进行可视化远程控制时，在虚拟机端打开 rviz2:

```
ros2 launch wheeltec_rviz2 wheeltec_rviz.launch.py
```

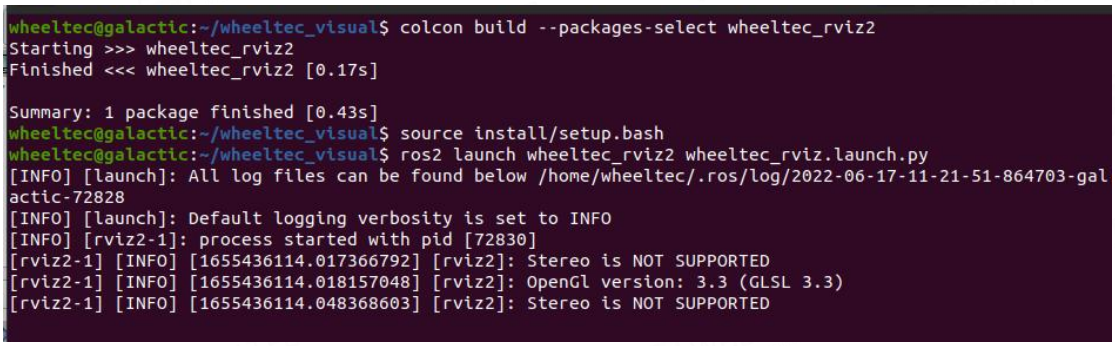

图 4-2-2　运行 wheeltec_rviz.launch.py

打开 rviz2 界面如图所示

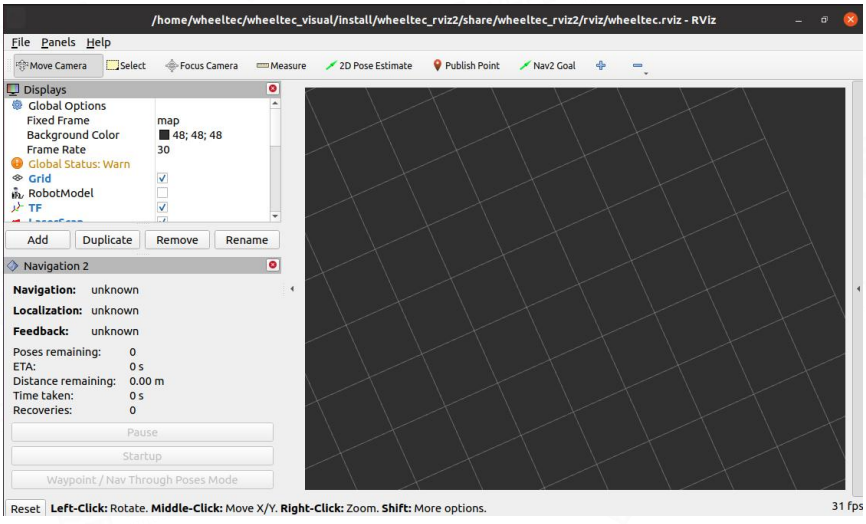

图 4-2-3　运行 rviz2 界面

rviz2 面板现在显示由 nav2_msgs/NavigateToPose 和

nav2_msgs/NavigateThroughPoses 操作发布的操作反馈。相对上一个版本(foxy)来说，在 Galactic 中用户可以直接通过 rviz 面板看到机器人到导航目标点的预计到达时间、距离目标的剩余距离、导航开始后经过的时间以及导航操作期间执行的恢复次数等信息。

② RTABMAPVIZ 可视化

WHEELTEC ROS2 更新的功能包含 RTABMAP 算法的建图导航，但目前在 RVIZ2 中无法加载 3D 点云：MapCloud plugin:Downloadmap still not working on ros2. 为了在调试过程中检查 3D 地图是否和建图过程中的地图一致，我们选择使用 RTABMAP 自带的可视化工具进行查看，即 RTABMAPVIZ.

首先需要确保虚拟机环境已经安装了 rtabmap:

```
sudo apt install ros-galactic-rtabmap*
```

```
wheeltec@galactic:~/wheeltec_visual$ sudo apt install ros-galactic-rtabmap*
[sudo] password for wheeltec:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'ros-galactic-rtabmap' for glob 'ros-galactic-rtabmap*'
Note, selecting 'ros-galactic-rtabmap-dbgsym' for glob 'ros-galactic-rtabmap*'
Note, selecting 'ros-galactic-rtabmap-ros-dbgsym' for glob 'ros-galactic-rtabmap*'
Note, selecting 'ros-galactic-rtabmap-ros' for glob 'ros-galactic-rtabmap*'
ros-galactic-rtabmap is already the newest version (0.20.18-1focal.20220514.021308).
ros-galactic-rtabmap-dbgsym is already the newest version (0.20.18-1focal.20220514.021308).
ros-galactic-rtabmap-ros is already the newest version (0.20.18-1focal.20220526.135235).
ros-galactic-rtabmap-ros-dbgsym is already the newest version (0.20.18-1focal.20220526.135235).
0 upgraded, 0 newly installed, 0 to remove and 111 not upgraded.
wheeltec@galactic:~/wheeltec_visual$
```

图 4-2-4　安装 ros-galactic-rtabmap

启动文件在 wheeltec_rviz2 功能包中，在工作空间路径下运行 RTABMAPVIZ：

```
source install/setup.bash
ros2 launch wheeltec_rviz2 wheeltec_rtabmapviz.launch.py
```

```
wheeltec@galactic:~/wheeltec_visual$ source install/setup.bash
wheeltec@galactic:~/wheeltec_visual$ ros2 launch wheeltec_rviz2 wheeltec_rtabmap
viz.launch.py
[INFO] [launch]: All log files can be found below /home/wheeltec/.ros/log/2022-0
6-17-16-11-31-294711-galactic-131793
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [rtabmapviz-1]: process started with pid [131795]
[rtabmapviz-1] RCUTILS_CONSOLE_STDOUT_LINE_BUFFERED is now ignored. Please set R
CUTILS_LOGGING_USE_STDOUT and RCUTILS_LOGGING_BUFFERED_STREAM to control the str
eam and the buffering of log messages.
[rtabmapviz-1] [INFO] [1655453492.162096332] [rtabmapviz]: rtabmapviz: Using con
figuration from "/home/wheeltec/.ros/rtabmapGUI.ini"
[rtabmapviz-1] libpng warning: iCCP: known incorrect sRGB profile
[rtabmapviz-1] libpng warning: iCCP: known incorrect sRGB profile
[rtabmapviz-1] libpng warning: iCCP: known incorrect sRGB profile
```

图 4-2-5　打开 rtabmapviz 可视化

打开 RTABMAPVIZ 可视化效果如图所示。

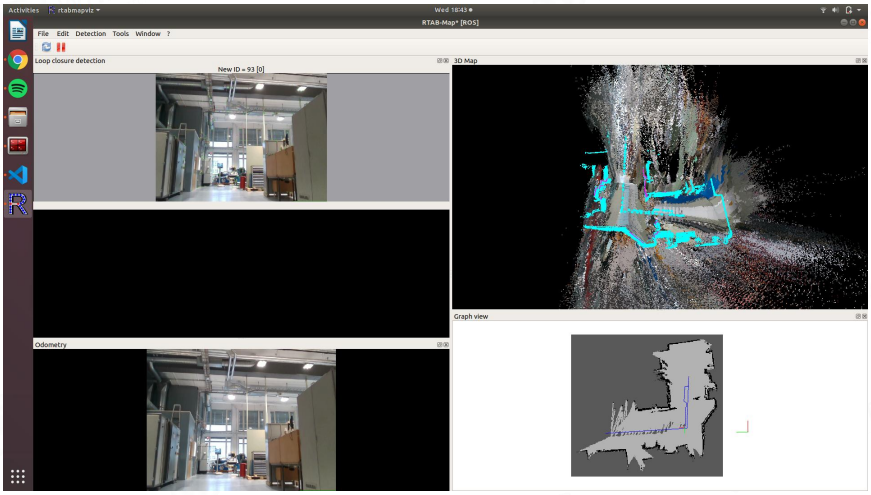

图 4-2-6　打开 rtabmapviz 可视化

③ ROS2 Tf Tree

目前 TF2 对 ROS 2 的支持只是初步的，有些部分 TF2 的功能显然还在进行中，tf2_frames 在 rqt 中尚不可用。若要查看 tf tree，则需要借助 tf2_tools 包。

首先确保系统中已安装 tf2-tools 功能包：

```
sudo apt install ros-galactic-tf2-tools
```

运行某个功能后查看 tf tree：

```
ros2 run tf2_tools view_frames
```

运行这句指令后需要稍微等待一小段时间，接着就会在当前运行指令的目录下生成两个文件。frames.pdf 和 frames.gv,这两个文件就是可视化的 tf 树。

④ ROS2 与 ROS1 通信

ROS2 的目标并不是取代 ROS1，所以会在很长的一段时间内，二者会一直并存。为了使用户能够使用 ROS1 和 ROS2 的功能包进行项目开发，可以通过 ros1_bridge 实现两个版本的连接与公用，该程序包提供了一个网桥，使 ROS 1 和 ROS 2 之间可以交换消息。

ros1_bridge 实际上是 ROS2 的一个功能包，用于自动或手动建立信息、话题和服务的映射，并在 ROS1 和 ROS2 之间进行通信。使用步骤示例：

1) 在 ROS1 中运行 roscore

```
source /opt/ros/noetic/setup.bash
roscore
```

2) 在 ROS1 的终端打开初始化底盘控制节点

```
roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch
```

3) 在 ROS2 的终端打开 ros1_bridge

```
source /opt/ros/galactic/setup.bash
ros2 run ros1_bridge dynamic_bridge
```

4) 在 ROS2 的终端打开键盘控制节点

```
ros2 run wheeltec_robot_keyboard wheeltec_keyboard
```

以上示例实现了速度话题/cmd_vel 从 ROS2 到 ROS1 的通信，如果想实现从 ROS1 到 ROS2 的通信的话，反过来也能完成。运行后的效果如图 3-3-1 所示。

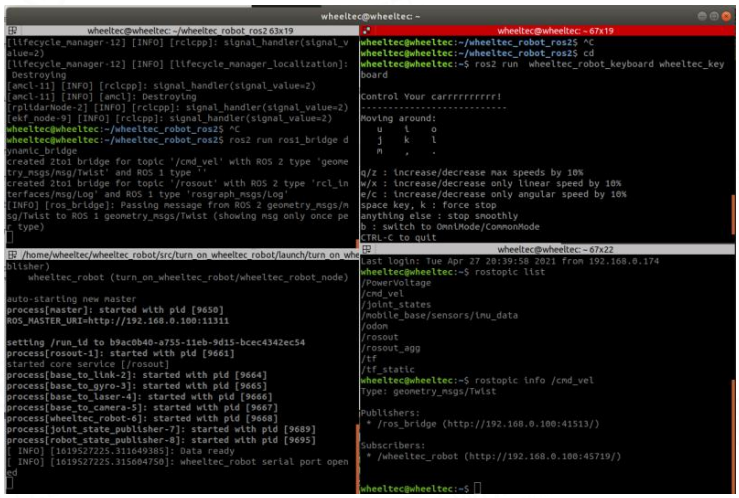

图 4-2-7　ROS1 与 ROS2 之间的服务通信

5. ROS2 中的 launch 文件与命令

5.1 ROS2 Launch 文件构成

ROS 2 启动文件可以用 Python、XML 和 YAML 编写。对于大多数应用程序，选择哪种 ROS 2 启动格式取决于开发人员的偏好。但是使用 Python 编写启动文件更具灵活性。具体体现在以下两个方面：

1. Python 是一种脚本语言，因此用户可以在启动文件中利用该语言及其库。
2. ros2/launch（一般启动功能）和 ros2/launch_ros（ROS 2 特定启动功能）是用 Python 编写的，因此用户对 XML 和 YAML 可能不会公开的启动功能具有较低级别的访问权限。

话虽如此，用 Python 编写的启动文件可能比用 XML 或 YAML 编写的启动文件更复杂和冗长。Python 编写的启动文件需要编译后才能启动。

ROS2 中 launch 文件通常以 LaunchName_launch.py 或 LaunchName.launch.py 命名，一般通过下面的命令启动：

```
ros2 launch <package_name> <launch_file_name>
```

或通过指定启动文件的路径直接运行文件

```
ros2 launch <path_to_launch_file>
```

① 常用 launch_ros.actions 介绍

要启动的节点或其他 launch 文件全部都传入 LaunchDescription()函数中，该函数中接受一个或多 launch.actions 或 launch_ros.actions 类型的对象，以下列举一下常用的 action，launch_ros.actions.x 一栏，如 Node 表示 launch_ros.actions.Node。

表 5-2-1 常用 launch_ros.action 含义

launch_ros.actions.x	功能注释
Node	启动一个 ros2 节点
PushRosNamespace	给一个节点或组设置命名空间
IncludeLaunchDescription	直接引用另一个 launch 文件
SetLaunchConfiguration	在 launch 文件内声明一个参数，并给定参数值
SetEnvironmentVariable	声明一个环境变量并给定环境变量的值
AppendEnvironmentVariable	对一个环境变量追加一个值，如果不存在则创建
TimerAction	在一段时间后执行一个或多个 action

ExecuteProcess	根据输入执行一个进程或脚本
DeclareLaunchArgument	声明一个启动描述参数，该参数具有名称、默认值和文档
GroupAction	将 action 分组，同组内的 action 可以统一设定参数方便集中管理
ExecuteProcess	根据输入执行一个进程或脚本
EmitEvent	发出一个事件，触发以注册的事件函数被调用
RegisterEventHandler	注册一个事件
UnregisterEventHandler	删除一个注册过的事件

② 常用 launch 文件写法

ros2 launch 文件中最主要的概念是 action，ros2 launch 把每一个要执行的节点，文件，脚本，功能等全部抽象成 action，用统一的接口来控制其启动，最主要的结构是：

```
def generate_launch_description():
    return LaunchDescription([
        action_1,
        action_2,
        action_n])
```

ROS2 中 python 格式的 launch 文件主要有两种常用的写法。一种在文件的前半部分配置好要运行的文件，用 add_action 方法加入即可；另一种是直接将启动的内容写在 return 语句。

第一种写法示例：

```
~/wheeltec_ros2/src/turn_on_wheeltec_robot/launch/wheeltec_lidar.launch.py
```

```
def generate_launch_description():
    lidar_dir = get_package_share_directory('lidar_ros2')
    lidar_launch_dir = os.path.join(lidar_dir, 'launch')

    Lsm10_dir = get_package_share_directory('lsm10_v2')
    Lsm10_launch_dir = os.path.join(Lsm10_dir, 'launch')

    Lsn10_dir = get_package_share_directory('lsn10')
    Lsn10_launch_dir = os.path.join(Lsn10_dir, 'launch')

    Ld14_dir = get_package_share_directory('ld14')
    Ld14_launch_dir = os.path.join(Ld14_dir, 'launch')

    Ld06_dir = get_package_share_directory('ldlidar_06_ros2')
    Ld06_launch_dir = os.path.join(Ld14_dir, 'launch')

    rplidar_ros = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(lidar_launch_dir, 'rplidar.launch.py')),)
    Lsn10 = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(Lsm10_launch_dir, 'ls_m10.launch.py')),)
    Lsn10 = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(Lsn10_launch_dir, 'ls_n10.launch.py')),)

    Ld14 = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(Ld14_launch_dir, 'ld14.launch.py')),)
    Ld06 = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(Ld06_launch_dir, 'ld06.launch.py')),)

    # Create the launch description and populate
    ld = LaunchDescription()
    #Select your radar here, options include:
    #rplidar_ros(A1、A2)、Lsm10、Lsn10、Ld14、Ld06
    ld.add_action(Ld14)
    return ld
```

图 5-1-1　wheeltec_lidar.launch.py 文件内容

第二种写法，将启动的内容写在 return 语句示例：

```
~/wheeltec_ros2/src/turn_on_wheeltec_robot/launch/base_serial.launch.py
```

```python
def generate_launch_description():
    akmcar = LaunchConfiguration('akmcar', default='false')
    return LaunchDescription([
        DeclareLaunchArgument(
            'akmcar',
            default_value='false',
            description='Use simulation (Gazebo) clock if true'),
#示例版本，非源码
#the default mode is not akm
        launch_ros.actions.Node(
            condition=UnlessCondition(akmcar),
            package='turn_on_wheeltec_robot',
            executable='wheeltec_robot_node',
            parameters=[{'usart_port_name': '/dev/wheeltec_controller',
                'serial_baud_rate': 115200,
                'robot_frame_id': 'base_footprint',
                'odom_frame_id': 'odom_combined',
                'cmd_vel': 'cmd_vel',
                'akm_cmd_vel': 'none',
                'product_number': 0,}],
            output='screen')
  ])
```

图 5-1-2　base_serial.launch.py 文件内容

5.2 编写 ROS2 launch 文件

以 turtlesim 功能包为基础，创建不通的 launch 文件。创建一个 wheeltec_launch 功能包，在该功能包中演示编写 ROS2 launch 文件。Wheeltec_launch 功能包在虚拟机中的路径为：/home/wheeltec/wheeltec_ros2/src/wheeltec_vmware

添加 launch 文件后需要编译功能包，在该功能包的 CMake 文件添加以下代码段，添加 launch 文件：

```
install(
  DIRECTORY launch
  DESTINATION share/${PROJECT_NAME})
```

添加后如图所示：

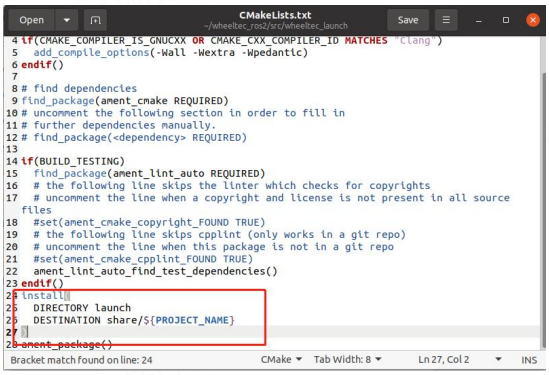

图 5-2-1　CMakeLists.txt 文件

在工作空间的目录下编译：

```
colcon build --packages-select wheeltec_launch
```

① 创建单节点的 launch 文件

进入 wheeltec_launch 功能包的 launch 文件夹下，创建一个 python 格式的

launch 文件，用于启动单个小海龟节点演示：

```
mkdir -p launch && cd launch
touch turtlesim_single_node.launch.py
```

在 turtlesim_single_node.launch.py 中添加以下内容：

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    turtle_node = Node(
            package='turtlesim',#功能包名
            executable='turtlesim_node',#功能包中可执行程序名
                )
    launch_description = LaunchDescription([turtle_node])#定义一个变量
    return launch_description
    #launch_description 作为调用 LaunchDescription 函数执行后的返回值
```

当使用命令行打开小海龟窗口时，指令如下：

```
ros2 run turtlesim turtlesim_node
```

该指令对应的 launch 内容是：

```
Node(
    package='turtlesim',
    executable='turtlesim_node',),
```

返回工作空间路径中编译后启动文件：

```
cd ~/wheeltec_ros2
colcon build --packages-select wheeltec_launch
source install/setup.bash
ros2 launch wheeltec_launch turtlesim_single_node.launch.py
```

此时启动了一个小海龟节点，如图所示。

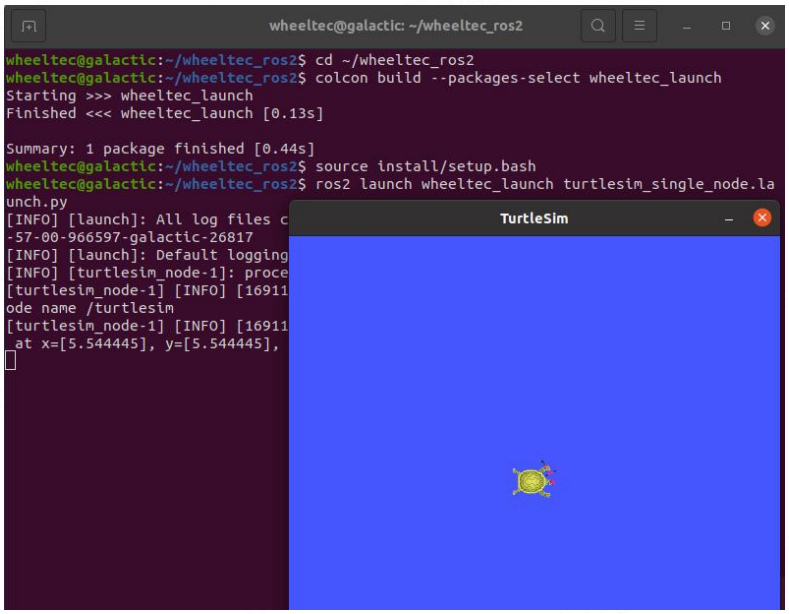

图 5-2-2　运行 turtlesim_single_node.launch.py 效果图

② 创建多节点的 launch 文件

进入 wheeltec_launch 功能包的 launch 文件夹下，创建一个 python 格式的 launch 文件，用于启动多个小海龟节点演示：

```
mkdir -p launch && cd launch
touch turtlesim_multi.launch.py
```

在 turtlesim_multi.launch.py 中添加以下内容，在该文件中增加了命名空间值。独特的命名空间允许系统启动两个没有节点名和主题名冲突的模拟器。

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    turtle_node = Node(
            package='turtlesim',
            namespace='turtlesim1',#命名空间 1
            executable='turtlesim_node',
        )
    turtle_node2=Node(
            package='turtlesim',
            namespace='turtlesim2',#命名空间 2
            executable='turtlesim_node',
        )
    launch_description = LaunchDescription([turtle_node,turtle_node2])#同时启动两个节点
    return launch_description
```

返回工作空间路径中编译后启动文件：

```
cd ~/wheeltec_ros2
colcon build --packages-select wheeltec_launch
source install/setup.bash
ros2 launch wheeltec_launch turtlesim_multi.launch.py
```

运行效果是启动了两个小海龟界面，查看节点列表：

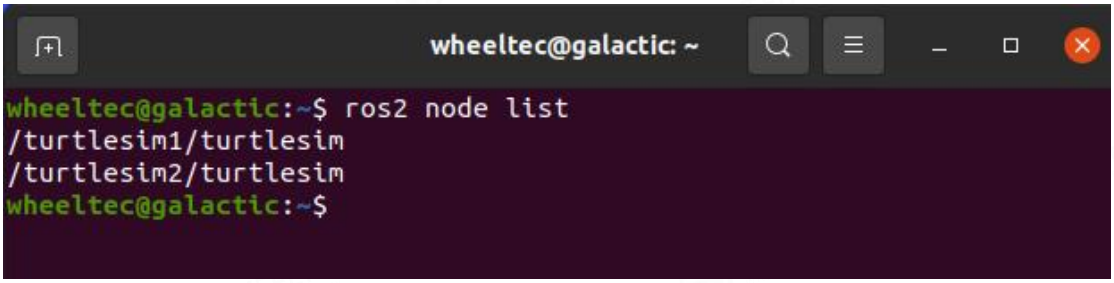

图 5-2-3　运行 turtlesim_multi_node.launch.py 时节点列表

③ 在 launch 文件映射话题名

进入 wheeltec_launch 功能包的 launch 文件夹下，创建一个 python 格式的 launch 文件，用于启动话题重命名演示：

```
mkdir -p launch && cd launch
touch turtlesim_remap_topic.launch.py
```

在 turtlesim_single_node.launch.py 中添加以下内容：

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    turtle_node = Node(
            package='turtlesim',
            executable='turtlesim_node',
            remappings=[("/turtle1/cmd_vel", "/cmd_vel")]#将/turtle1/cmd_vel 重
映射为/cmd_vel
                )
    launch_description = LaunchDescription([turtle_node])
    return launch_description
```

映射前节点发布/turtle1/cmd_vel 话题，设置重映射后该节点将发布/cmd_vel 的话题。

④ launch 文件嵌套

在一个 launch 文件中启动另一个 launch 文件，进入 wheeltec_launch 功能包的 launch 文件夹下，创建一个 python 格式的 launch 文件，用于启动文件嵌套演示：

```
mkdir -p launch && cd launch
touch turtlesim_include.launch.py
```

在 turtlesim_include.launch.py 中添加以下内容：

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

def generate_launch_description():
    turtle_node1 = IncludeLaunchDescription(
#wheeltec_launch 为嵌套的功能包名，'launch'为嵌套的文件所在功能包中的文件夹名
    PythonLaunchDescriptionSource([os.path.join(get_package_share_directory('wh
eeltec_launch'), 'launch',
    'turtlesim_multi_node.launch.py')])#turtlesim_multi_node.launch.py 为要包含
的启动文件
    )
```

```
    ld = LaunchDescription()
    ld.add_action(turtle_node1)
    return ld
```

返回工作空间路径中编译后启动文件：

```
cd ~/wheeltec_ros2
colcon build --packages-select wheeltec_launch
source install/setup.bash
ros2 launch wheeltec_launch turtlesim_include.launch.py
```

运行效果是启动 turtlesim_multi_node.launch.py 文件，即启动了两个小海龟界面，查看节点列表：

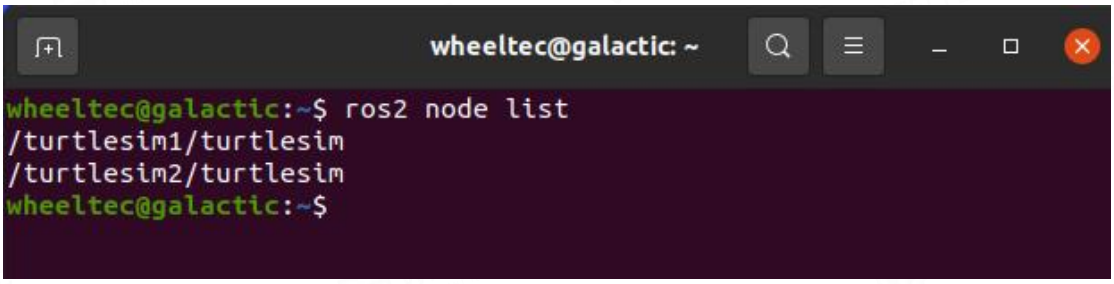

图 5-2-4　运行 turtlesim_include_node.launch.py 时节点列表

6. ROS2 功能包调试前期准备

前期准备包括车型切换与传感器切换。目前在 WHEELTEC ROS2 小车中用户可能需要切换的传感有相机和雷达。若用户使用的传感器与镜像默认传感器不一致，则需要按照此教程对程序进行修改。

6.1 车型切换

每个人使用的 WHEELTEC 机器人型号不一样，但使用的 ROS2 镜像是一样的，需要手动修改车型配置文件。其中包括 turn_on_wheeltec_robot.launch.py、robot_mode_description.launch.py、以及 wheeltec_robot_nav2 功能包中的导航参数文件。参数不同会导致导航效果不同。

① **选择小车 urdf 模型**

出厂配置默认是 mini_mec 小车的模型，用户若使用其它车型的小车，如 mini_akm 小车，需要在文件中修改对应的小车模型，修改文件：

`wheeltec_ros2/src/turn_on_wheeltec_robot/launch/turn_on_wheeltec_robot.launch.py`

如果用户使用的是服务机器人或 mini 版小车如 mini_diff（mini 差速小车），则修改对应的 minibot_type，修改图示参数为 mini_diff，启动 ld.add_action(minibot_type)并屏蔽 ld.add_action(flagship_type)，如图所示。

```
#select a robot model,the default model is mini_mec
#minibot.launch.py contains commonly used robot models
#launch_arguments choices:mini_mec、mini_akm、mini_tank、mini_4wd、mini_diff、mini_omni、brushless_senior_diff
#!!!At the same time, you need to modify ld.add_action(minibot_type) and #ld.add_action(flagship_type)
minibot_type = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(launch_dir, 'robot_mode_description_minibot.launch.py')),
        launch_arguments={'mini_diff': 'true'}.items(),
)                                                                    修改车型

#robot_mode_description.launch.py contains flagship products, usually larger chassis robots
#launch_arguments choices:

#senior_akm、top_akm_bs、top_akm_dl
#senior_mec_bs、senior_mec_dl、 top_mec_bs、 top_mec_dl、 mec_EightDrive_robot、 flagship_mec_bs_robot、 flagship_mec_dl_robot、
#senior_omni、top_omni
#senior_4wd_bs_robot、senior_4wd_dl_robot、flagship_4wd_bs_robot、flagship_4wd_dl_robot、top_4wd_bs_robot、top_4wd_dl_robot
#senior_diff_robot、four_wheel_diff_bs、four_wheel_diff_dl、flagship_four_wheel_diff_bs_robot、flagship_four_wheel_diff_dl_robot
#!!!At the same time, you need to modify ld.add_action(flagship_type) and #ld.add_action(minibot_type)
flagship_type = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(launch_dir, 'robot_mode_description.launch.py')),
        launch_arguments={'top_akm_dl': 'true'}.items(),
)

ld = LaunchDescription()

ld.add_action(minibot_type)
#ld.add_action(flagship_type)          flagship_type保持屏蔽
ld.add_action(carto_slam_dec)
ld.add_action(wheeltec_robot)
ld.add_action(base_to_link)
ld.add_action(base_to_gyro)
ld.add_action(joint_state_publisher_node)
ld.add_action(imu_filter_node)
ld.add_action(robot_ekf)

return ld
```

如果用户使用的是非 mini 小车类型如高配阿克曼机器人，则修改对应的 flagship_type，修改图示参数为 senior_akm，启动 ld.add_action(flagship_type)并屏蔽 ld.add_action(minibot_type)，如图所示。

```
#robot_mode_description.launch.py contains flagship products, usually larger chassis robots
#launch_arguments choices:

#senior_akm、top_akm_bs、top_akm_dl
#senior_mec_bs、senior_mec_dl、 top_mec_bs、 top_mec_dl、 mec_EightDrive_robot、 flagship_mec_bs_robot、 flagship_mec_dl_robot、
#senior_omni、top_omni
#senior_4wd_bs_robot、senior_4wd_dl_robot、flagship_4wd_bs_robot、flagship_4wd_dl_robot、top_4wd_bs_robot、top_4wd_dl_robot
#senior_diff_robot、four_wheel_diff_bs、four_wheel_diff_dl、flagship_four_wheel_diff_bs_robot、flagship_four_wheel_diff_dl_robot
#!!!At the same time, you need to modify ld.add_action(flagship_type) and #ld.add_action(minibot_type)
flagship_type = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(launch_dir, 'robot_mode_description.launch.py')),
        launch_arguments={'senior_akm': 'true'}.items(),
)

ld = LaunchDescription()

#ld.add_action(minibot_type)
ld.add_action(flagship_type)
ld.add_action(carto_slam_dec)
ld.add_action(wheeltec_robot)
ld.add_action(base_to_link)
ld.add_action(base_to_gyro)
ld.add_action(joint_state_publisher_node)
ld.add_action(imu_filter_node)
ld.add_action(robot_ekf)
```

minibot_type保持屏蔽

图 6-1-2　turn_on_wheeltec_robot.launch.py 文件

修改完后需要编译使其生效：

```
colcon build --packages-select turn_on_wheeltec_robot
```

启动底盘文件，终端 robot_state_publisher 节点打印的节点名可以确认用户目前使用的车型。

```
wheeltec@wheeltec:~/wheeltec_ros2$ ros2 launch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch.py
[INFO] [launch]: All log files can be found below /home/wheeltec/.ros/log/2024-04-28-14-34-30-856320-wheeltec-27656
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [robot_state_publisher-1]: process started with pid [27657]
[INFO] [static_transform_publisher-2]: process started with pid [27659]
[INFO] [static_transform_publisher-3]: process started with pid [27661]
[INFO] [wheeltec_robot_node-4]: process started with pid [27663]
[INFO] [static_transform_publisher-5]: process started with pid [27665]
[INFO] [static_transform_publisher-6]: process started with pid [27667]
[INFO] [joint_state_publisher-7]: process started with pid [27682]
[INFO] [imu_filter_madgwick_node-8]: process started with pid [27713]
[INFO] [ekf_node-9]: process started with pid [27721]
[static_transform_publisher-2] [WARN] [1714286072.353426468] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [WARN] [1714286072.351081376] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [INFO] [1714286072.386792493] [base_to_camera_link]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('0.000000', '0.000000', '0.250000')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_footprint' to 'camera_link'
[static_transform_publisher-5] [WARN] [1714286072.355573762] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-6] [WARN] [1714286072.384818291] []: Old-style arguments are deprecated; see --help for new-style arguments
[robot_state_publisher-1] [WARN] [1714286072.402683354] [robot_state_publisher_mini_diff]: No robot_description parameter, but command-line argument available.  Assuming argument is name of URDF file.  This backwards compatibility fallback will be removed in the future.
[wheeltec_robot_node-4] [INFO] [1714286072.410349209] [wheeltec_robot]: wheeltec_robot Data ready
[wheeltec_robot_node-4] [INFO] [1714286072.416359527] [wheeltec_robot]: wheeltec_robot serial port opened
[static_transform_publisher-2] [INFO] [1714286072.430209873] [base_to_laser]: Spinning until stopped - publishing transform
[static_transform_publisher-2] translation: ('0.048000', '0.000000', '0.155000')
[static_transform_publisher-2] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-2] from 'base_footprint' to 'laser'
[robot_state_publisher-1] [INFO] [1714286072.442466237] [robot_state_publisher_mini_diff]: got segment base_link
[robot_state_publisher-1] [INFO] [1714286072.442571648] [robot_state_publisher_mini_diff]: got segment front_wheel_link
[robot_state_publisher-1] [INFO] [1714286072.442589249] [robot_state_publisher_mini_diff]: got segment left_wheel_link
[robot_state_publisher-1] [INFO] [1714286072.442599137] [robot_state_publisher_mini_diff]: got segment right_wheel_link
[static_transform_publisher-5] [INFO] [1714286072.451581038] [base_to_link]: Spinning until stopped - publishing transform
[static_transform_publisher-5] translation: ('0.000000', '0.000000', '0.000000')
[static_transform_publisher-5] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-5] from 'base_footprint' to 'base_link'
[static_transform_publisher-6] [INFO] [1714286072.469613937] [base_to_gyro]: Spinning until stopped - publishing transform
[static_transform_publisher-6] translation: ('0.000000', '0.000000', '0.000000')
[static_transform_publisher-6] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-6] from 'base_footprint' to 'gyro_link'
[imu_filter_madgwick_node-8] [INFO] [1714286072.520491383] [imu_filter_madgwick]: Starting ImuFilter
[imu_filter_madgwick_node-8] [INFO] [1714286072.523356298] [imu_filter_madgwick]: Using dt computed from message headers
[imu_filter_madgwick_node-8] [INFO] [1714286072.523412171] [imu_filter_madgwick]: The gravity vector is kept in the IMU message.
[imu_filter_madgwick_node-8] [INFO] [1714286072.525335779] [imu_filter_madgwick]: Imu filter gain set to 0.100000
[imu_filter_madgwick_node-8] [INFO] [1714286072.525390821] [imu_filter_madgwick]: Gyro drift bias set to 0.000000
[imu_filter_madgwick_node-8] [INFO] [1714286072.525419174] [imu_filter_madgwick]: Magnetometer bias values: 0.000000 0.000000 0.000000
[imu_filter_madgwick_node-8] [INFO] [1714286072.550289944] [imu_filter_madgwick]: First IMU message received.
[joint_state_publisher-7] [INFO] [1714286073.153747305] [joint_state_publisher]: Waiting for robot_description to be published on the robot_description topic ...
```

② 选择导航的参数文件

不同小车模型对应的外观参数以及膨胀半径不一致，阿克曼模型的小车还会涉及到最小转弯半径的参数，所以在调试导航前请确认导航参数与小车模型是否匹配。修改文件中的 my_param_file 参数：

```
/home/wheeltec/wheeltec_ros2/src/wheeltec_robot_nav2/launch/
wheeltec_nav2.launch.py
```

my_param_file 参数文件名可以在注释查询，如 mini_akm 小车，将 my_param_file 的值改为 param_mini_akm.yaml 即可。修改后的文件如图所示。

```
#Modify the model parameter file, the options are:
#param_mini_akm.yaml、param_mini_4wd.yaml、param_mini_diff.yaml、
#param_mini_mec.yaml、param_mini_omni.yaml、param_mini_tank.yaml、
#param_senior_akm.yaml、param_senior_diff.yaml、param_senior_mec_bs.yaml
#param_senior_mec_dl.yaml、param_top_4wd_bs.yaml、param_top_4wd_dl.yaml
#param_top_akm_dl.yaml、param_four_wheel_diff_dl.yaml、param_four_wheel_diff_bs.yaml
#If you want to use the DWB algorithm, you can change the input parameters to wheeltec-dwb.yaml
my_param_dir = os.path.join(my_nav_dir, 'param','wheeltec_param')
my_param_file = 'param_mini_akm.yaml'
```

图 6-1-3　修改 wheeltec_nav2.launch.py 文件

③ 膨胀半径

膨胀半径即围绕致命障碍物膨胀成本地图的半径。不同用户的导航环境不一样，导航时可能需要修改膨胀半径这一参数。以 mini_akm 小车为例，修改文件~/wheeltec_ros2/src/wheeltec_robot_nav2/param/wheeltec_param/param_mini_akm.yaml，在文件中搜索 “inflation_radius”这一参数，膨胀半径单位默认为米，用户可以按需修改。

```
163  local_costmap:
164    local_costmap:
165      ros__parameters:
166        update_frequency: 5.0
167        publish_frequency: 2.0
168        global_frame: odom_combined
169        robot_base_frame: base_footprint
170        use_sim_time: False
171        rolling_window: true
172        width: 3
173        height: 3
174        resolution: 0.05
175        footprint: "[ [-0.1350, -0.1110], [-0.1350, 0.1110], [0.1350, 0.1110], [0.1350, -0.1110] ]"
176        plugins: ["obstacle_layer", "inflation_layer"]
177        inflation_layer:
178          plugin: "nav2_costmap_2d::InflationLayer"
179          cost_scaling_factor: 3.0
180          inflation_radius: 0.15
```

图 6-1-4　修改 wheeltec_nav2.launch.py 文件

④ 修改完文件后记得编译

和 ROS1 不同的是，ROS2 中修改 launch 文件需要编译后才能生效。确认以上三个修改车型的步骤操作完成后，进入工作空间路径下编译：

```
cd ~/wheeltec_ros2
colcon build --packages-select turn_on_wheeltec_robot
colcon build --packages-select wheeltec_nav2
```

6.2 相机切换

WHEELTEC ROS2 小车目前支持 AstraS、Astra Pro、Daibai、Daibai_Pro、Gemini_Pro 相机和 Realsense 相机。程序中默认使用 Astra S 相机，若用户使用的是其他相机，则可以按照以下步骤更改程序。

打开相机指令：

```
ros2 launch turn_on_wheeltec_robot wheeltec_camera.launch.py
```

修改相机型号，修改文件

```
~/wheeltec_ros2/src/turn_on_wheeltec_robot/launch/wheeltec_camera.launch.py
```

其中可选参数为：Astra_S、Astra_Pro、Dabai、Gemini、Wheeltec_Usbcam。如用户使用 AstraS 相机，则修改后的 wheeltec_camera.launch.py 文件如下所示。

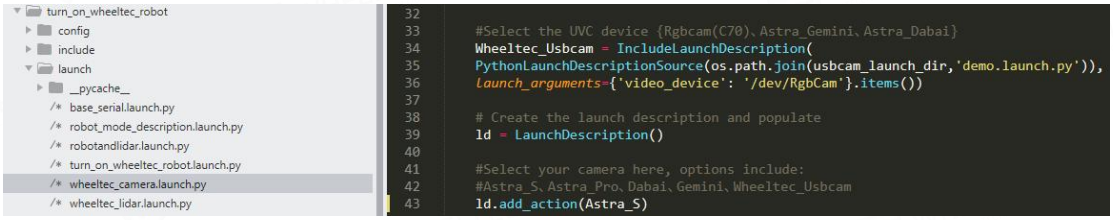

图 6-2-1　修改 wheeltec_camera.launch.py 文件

修改文件后进行编译：

```
cd ~/wheeltec_ros2
colcon build --packages-select turn_on_wheeltec_robot
```

注意：usbcam 功能包为了适配不同的相机和别名规则，外置了参数接口（默认接口为 WHEELTEC C70 相机的串口别名：/dev/Rgbcam），可自行根据需求修改。

6.3 雷达切换

WHEELTEC ROS2 小车除了支持常用的思岚雷达之外，还支持镭神的 M10、N10(网口和串口版)雷达，和乐动雷达 LD14、LD06、LD19 雷达。默认使用思岚的 A1 雷达。若用户使用其它款式雷达，需要在此文件夹下进行更换：

```
wheeltec_ros2/src/turn_on_wheeltec_robot/launch/wheeltec_lidar.launch.py
```

在文件最后一行修改参数，可选参数为：Lsm10P、Lsm10、Lsn10P、Lsn10、Ld14、Ld06。

若使用镭神 LSN10 雷达，则修改参数后文件内容如下：

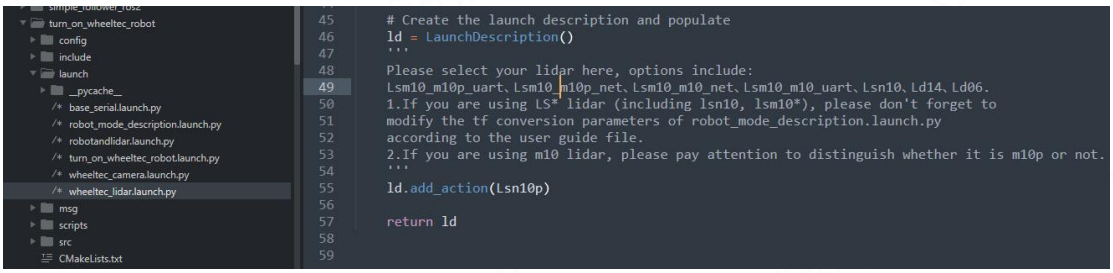

图 6-3-1　修改 wheeltec_lidar.launch.py 文件

修改文件后进行编译：

```
cd ~/wheeltec_ros2
```

```
colcon build --packages-select turn_on_wheeltec_robot
```

① 使用镭神雷达注意事项

使用 LSm10、LSn10 除了要修改 wheeltec_lidar.launch.py 文件之外，还需要修改 TF 静态转换的参数，因为 LSm10*和 LSn10*雷达扫描的方向与常用雷达相反，所以需要在 TF 静态转换中把方向调转。

修改文件：

```
wheeltec_ros2/src/turn_on_wheeltec_robot/launch/ robot_mode_description.launch.py
```

选择对应的车型 GroupAction，修改 base_to_laser 节点的 x 方向参数，以 mini_akm 小车为例，则在此处将 3.14 改为 0：

图 6-3-2　修改 robot_mode_description.launch.py 文件

修改文件后进行编译：

```
cd ~/wheeltec_ros2
colcon build --packages-select turn_on_wheeltec_robot
```

② 雷达角度屏蔽

不同雷达角度屏蔽修改的文件不同。WHEELTEC ROS2 中雷达文件均在~/wheeltec_ros2/src/wheeltec_lidar_ros2 目录下。在该目录下，LDlidar 表示乐动雷达，LSlidar 表示镭神雷达。

乐动雷达角度屏蔽只需要修改雷达启动文件即可，乐动 LD06 雷达修改角度屏蔽步骤，修改文件

```
~/wheeltec_ros2/src/wheeltec_lidar_ros2/LDlidar/ldlidar06/launch/ld06.launch.py
```

如要屏蔽乐动 LD06 雷达扫描的角度为[135，225]这一区间，文件内容如图所示。

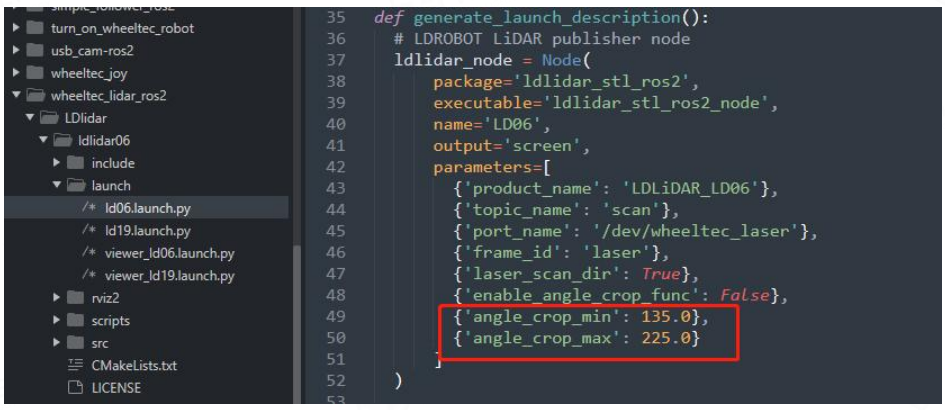

图 6-3-3　修改 ld06.launch.py 文件

文件修改后需要编译

```
cd ~/wheeltec_ros2
colcon build --packages-select ldlidar_stl_ros2
```

乐动 LD14 雷达修改角度屏蔽步骤，修改文件

```
~/wheeltec_ros2/src/wheeltec_lidar_ros2/LDlidar/ldlidar14/launch/ld14.launch.py
```

如要屏蔽乐动 LD14 雷达扫描的角度为[135，225]这一区间，文件内容如图所示。

图 6-3-4　修改 ld14.launch.py 文件

文件修改后需要编译

```
cd ~/wheeltec_ros2
colcon build --packages-select ldlidar_sl_ros2
```

使用镭神雷达进行角度屏蔽，需要修改雷达对应的参数文件。镭神雷达包括 M10（串口版和网口版）、M10P（串口版和网口版）、N10 这几款雷达。雷达文件结构如图所示。

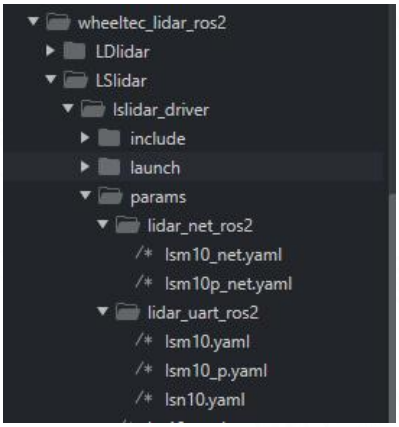

图 6-3-5　LSlidar 文件框架

不同雷达使用的参数文件不同，如 LSN10 雷达修改屏蔽角度，需要修改文件

```
~/wheeltec_ros2/src/wheeltec_lidar_ros2/LSlidar/lslidar_driver/params/lidar_uart_ros2/lsn10.yaml
```

如要屏蔽镭神 LSN10 雷达扫描的角度为[135，225]这一区间，文件内容如图所示。

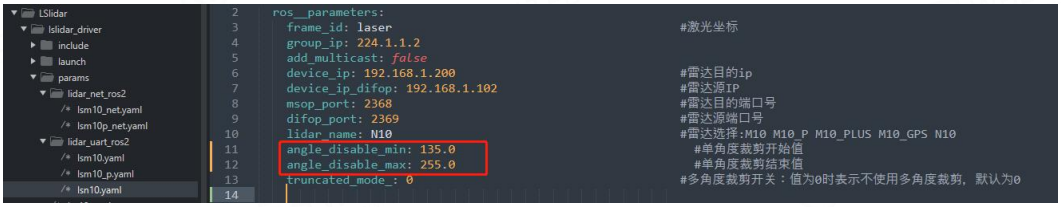

图 6-3-6　lsn10.yaml 文件内容

如 LSM10 串口版雷达修改屏蔽角度，需要修改文件

```
~/wheeltec_ros2/src/wheeltec_lidar_ros2/LSlidar/lslidar_driver/params/lidar_uart_ros2/lsm10.yaml
```

如要屏蔽镭神 LSM10 串口版雷达扫描的角度为[135，225]这一区间，文件内容如图所示。

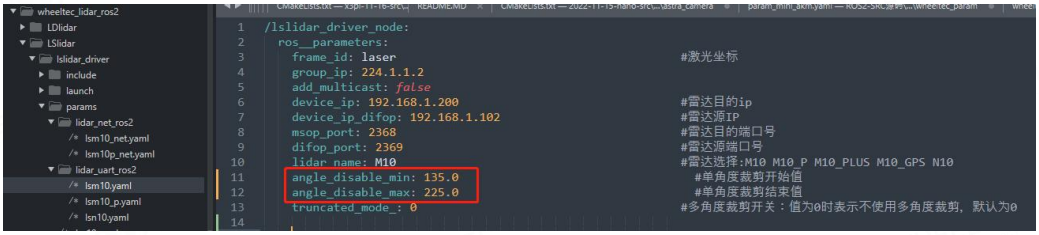

图 6-3-7　lsm10.yaml 文件内容

如 LSM10P 网口版雷达修改屏蔽角度，需要修改文件

```
~/wheeltec_ros2/src/wheeltec_lidar_ros2/LSlidar/lslidar_driver/params/lidar_net_ros2/lsm10p_net.yaml
```

如要屏蔽镭神 LSM10P 网口版雷达扫描的角度为[135，225]这一区间，文件内容如图所示。

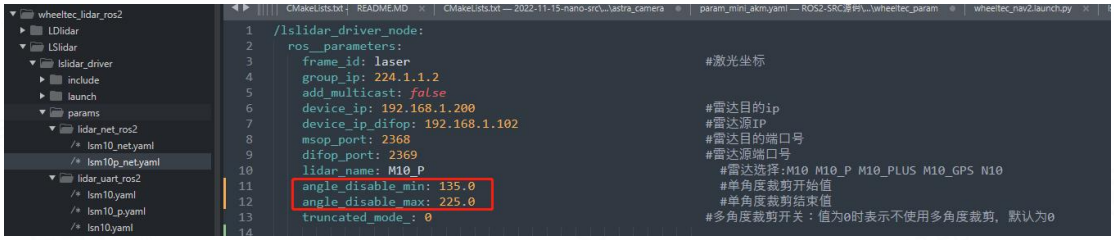

图 6-3-8　lsm10p_net.yaml 文件内容

文件修改后需要编译

```
cd ~/wheeltec_ros2
colcon build --packages-select lslidar_driver
```

7. ROS2 功能包调试教程

7.1 Turn_on_wheeltec_robot[机器人底层驱动]

turn_on_wheeltec_robot 功能包的启动文件主要有：

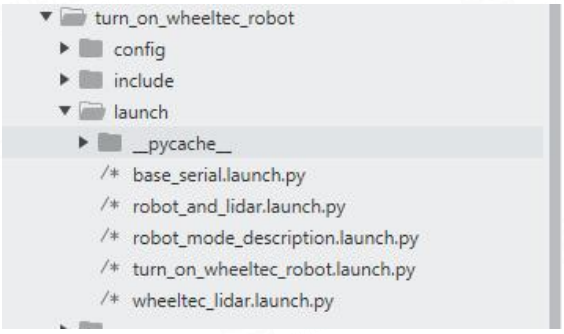

图 7-1-1　turn_on_wheeltec_robot 功能包

其中 base_serial.launch.py 是启动底盘控制节点，实现 ROS 主控与 STM32 的通信；

robot_and_lidar.launch.py 文件是打开底盘控制和雷达节点，主要用于验证传感器是否正常；

robot_mode_description.launch.py 用于导入小车的 urdf 模型，包括传感器和机器人底层的静态 tf 转换；

wheeltec_lidar.launch.py 为 WHEELTEC ROS2 小车的雷达控制节点，用户在这个文件中切换雷达型号；

turn_on_wheeltec_robot.launch.py 包含底盘控制、传感器与 base_footprint 的 TF 静态转换、机器人描述文件以及融合滤波的功能。

turn_on_wheeltec_robot 功能包主要用于启动底盘，运行指令为：

```
ros2 launch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch.py
```

正常运行如图所示：

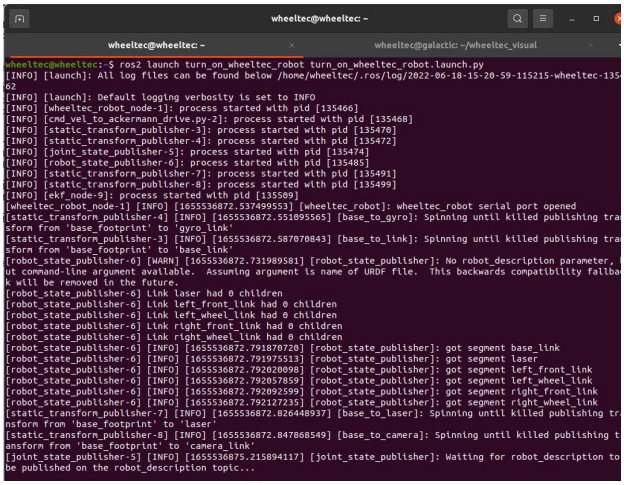

图 7-1-2　运行 turn_on_wheeltec_robot.launch.py

此时打开 RVIZ2 可以看到小车的 urdf 模型：

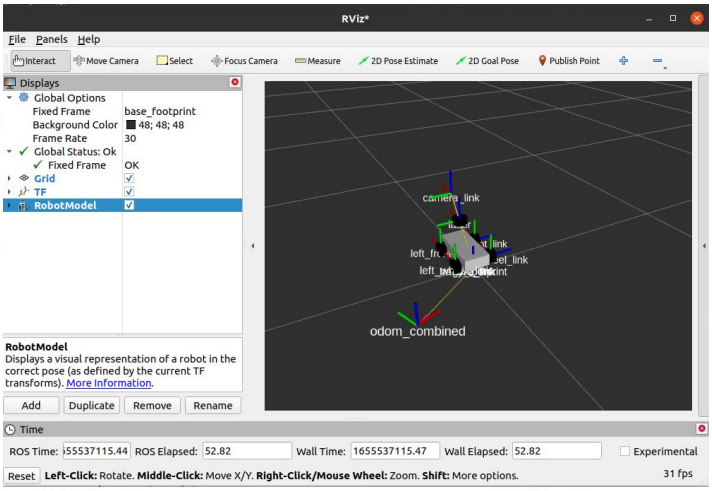

图 7-1-3　rviz2 中的机器人模型

如果 rviz 没有显示模型，需要检查 RobotModel 中的 Description 有没有选中话题，话题为/robot_description。

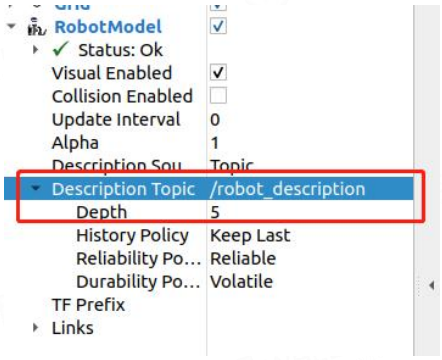

图 7-1-4　rviz2 配置

7.2 Wheeltec_robot_slam[2D 建图]

wheeltec_robot_slam 功能包主要用于 WHEELTEC ROS2 小车进行 2D 建图 gmapping 建图算法、slam_toolbox 建图算法和 cartographer 建图算法。运行建图时可以使用蓝牙/航模/键盘控制节点/USB 手柄控制小车运动。

① gmapping 建图

slamgapping 软件包包含用于 OpenSlam 的 Gmapping 的 ROS 包装器，它提供了基于激光的 SLAM（同时定位和映射），作为 slam_gmapping 的 ROS 节点。

使用 Gmapping 算法进行 2D 建图，指令如下。建图页面如图 3-2-3 所示。

```
ros2 launch slam_gmapping slam_gmapping.launch.py
```

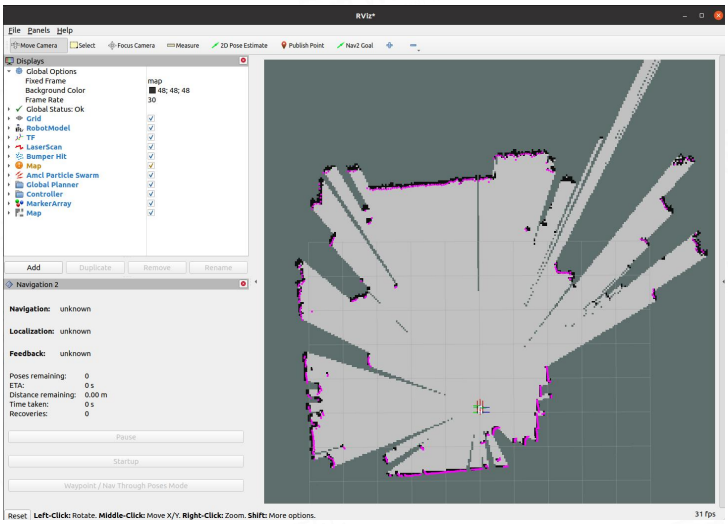

图 7-2-1　gmapping 建图

② slam_toolbox 建图

slam_toolbox 是 2D SLAM 构建的一组工具和功能，该软件包将允许用户完全序列化要重新加载的 SLAM 映射的数据和姿态图，以继续映射，本地化，合并或进行其他操作。关于 slam_toolbox 算法更多详细的介绍可以这里找到。

使用 slam_toolbox 进行 2D 建图，指令如下：

```
ros2 launch wheeltec_slam_toolbox online_async.launch.py
```

建图界面如图所示

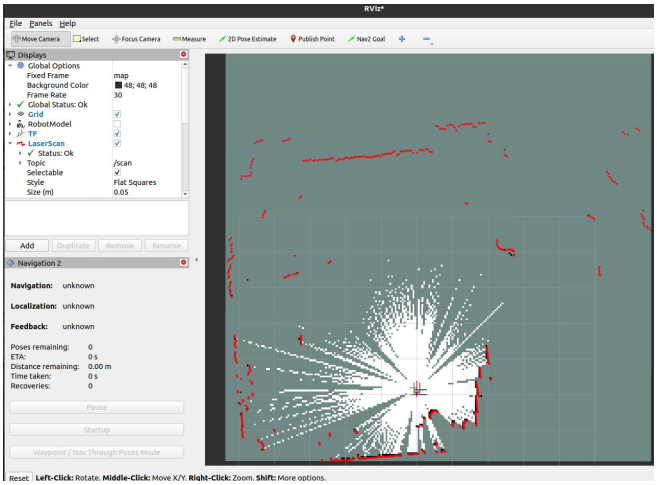

图 7-2-2　slam_toolbox 建图

③ slam_toolbox 工具保存地图

使用 slam_toolbox 建图时，系统可以调用 slam_toolbox 功能包中的 rviz2 插件，实现在 rviz2 中即时保存地图。注意：只针对 slam_toolbox 建图算法。主要步骤为：

Step1：运行 slam_toolbox 建图

```
ros2 launch wheeltec_slam_toolbox online_async.launch.py
```

Step2：打开 rviz2，点击左上角的 Panels 添加面板，点击 Add New Panel 选项。

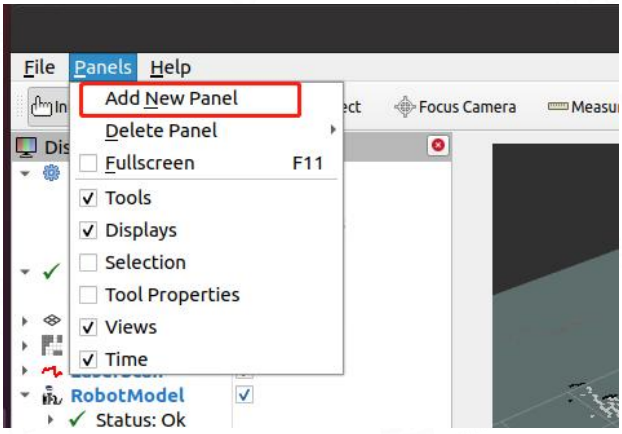

图 7-2-3　添加新的面板(1)

在弹窗中选择 slam_toolbox 一栏下的 SlamToolboxPlugin 插件

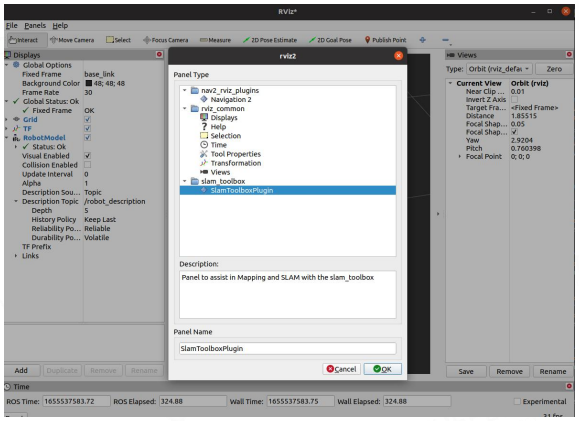

图 7-2-4　添加新的面板(2)

点击 ok，在 rviz2 中出现新的面板，界面如图所示：

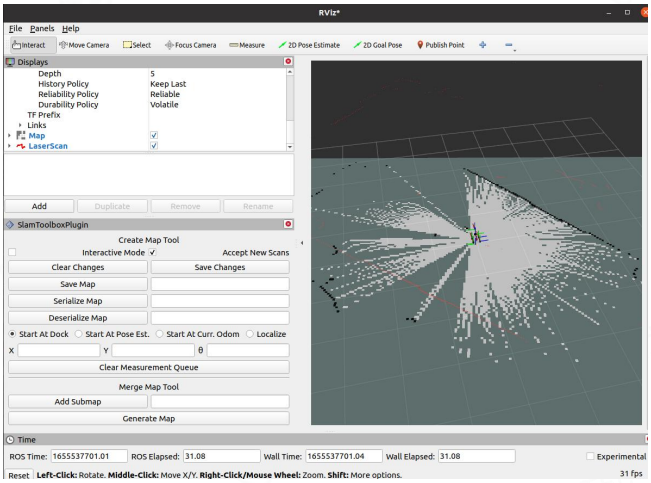

图 7-2-5　添加新的面板(3)

点击 Save Map 即可保存 slam_toolbox 建图，运行 slam_toolbox 建图的终端提示如下：

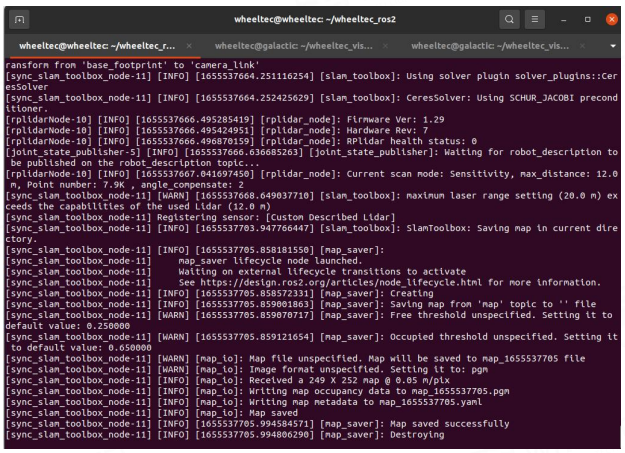

图 7-2-6　保存图片时终端输出

地图保存在运行 slam_toolbox 建图算法的终端所在目录中。

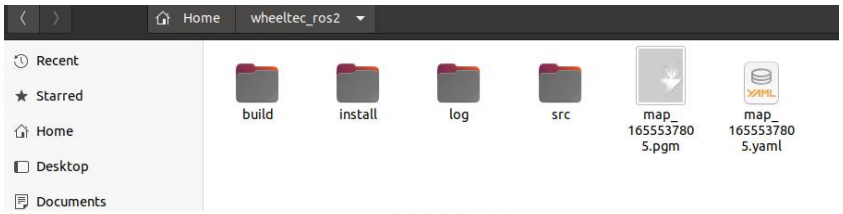

图 7-2-7　地图保存结果

④ cartographer 建图

Cartographer 是一个跨多个平台和传感器配置提供 2D 和 3D 实时同步定位和映射 (SLAM) 的系统。出厂镜像中已安装好 cartographer 建图算法，用户自行安装请运行以下指令：

```
sudo apt install ros-humble-cartographer*
```

源码仓库：https://github.com/ros2/cartographer_ros

官方文档：https://google-cartographer.readthedocs.io/en/latest/

在 wheeltec robot 中打开 Cartographer 建图：

```
ros2 launch wheeltec_cartographer cartographer.launch.py
```

建图效果如图所示：

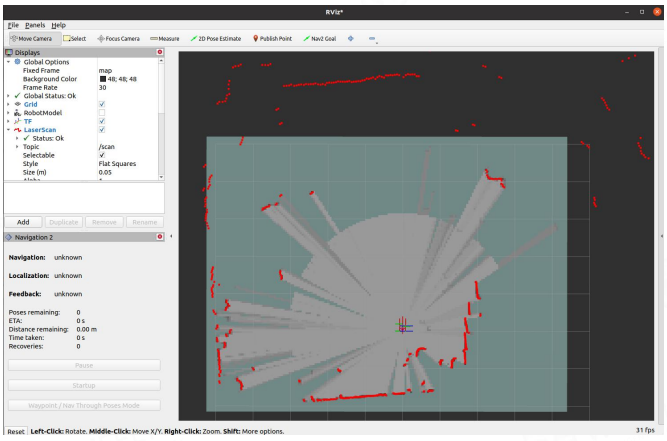

图 7-2-9　cartographer 建图

⑤ 保存地图

Map Server 用于处理堆栈的地图加载请求并托管地图主题的服务器，同时它也是一个地图保存服务器，根据服务请求保存地图。存在一个类似于 ROS 1 的地图保护程序 CLI，用于单个地图保存。

WHEELTEC ROS2 机器人保存地图指令为：

```
ros2 launch wheeltec_nav2 save_map.launch.py
```

运行 save_map.launch.py 文件调用 Map Server 服务，将地图保存在 ~/wheeltec_ros2/src/wheeltec_robot_nav2/map 目录下。

建图完成后运行保存地图指令示例：

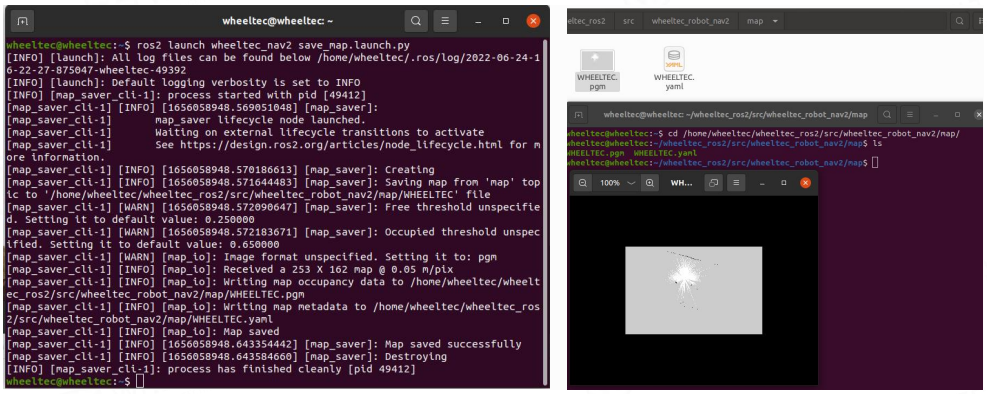

图 7-2-10　保存地图

此时可以在~/wheeltec_ros2/src/wheeltec_robot_nav2/map 目录下看到地图文件 WHEELTEC.pgm 和 WHEELTEC.yaml，双击可查看地图图像。

指令保存地图：

```
ros2 run nav2_map_server map_saver_cli -f <map_dir>/<map_name>
```

7.3 Wheeltec_robot_rrt2[RRT Exploration]

wheeltec_robot_rrt2 基于快速探索随机树 (RRT) 算法实现 WHEELTEC 机器人的自主探索建图功能。RRT（Rapidly-Exploring Random Trees）即快速随机扩展树，它是一种单一查询算法，一边抽样一边建图，通过不断随机探索来实现遍历。RRT 算法概率完备，可用于进行路径规划与地图边界探索。但因为其随机性比较强，所规划的路径并不一定是最优解。在 WHEELTEC ROS2 机器人中 RRT 算法通过将地图边界点作为目标点来实现小车自主导航探索，同时开启建图节点，实现自主建图。

在 WHEELTEC ROS2 机器人中只用到了 rrt_exploration 的全局 RRT 边界点检测器节点、本地 RRT 边界点检测器节点、过滤节点和分配者节点。

① 修改 navigation 导航参数

RRT Exploration 自主探索建图会一边导航一边建图，但不同的车型导航参数不一致，膨胀半径也不同，默认膨胀半径 inflation_radius 的值为 0.55m。用户根据自己的车型修改改参数，需要修改的文件为：

```
~/wheeltec_ros2/src/wheeltec_robot_rrt2/config/nav2_params.yaml
```

如 mini 系列小车修改 inflation_radius 值为 0.25， 文件修改后内容如下

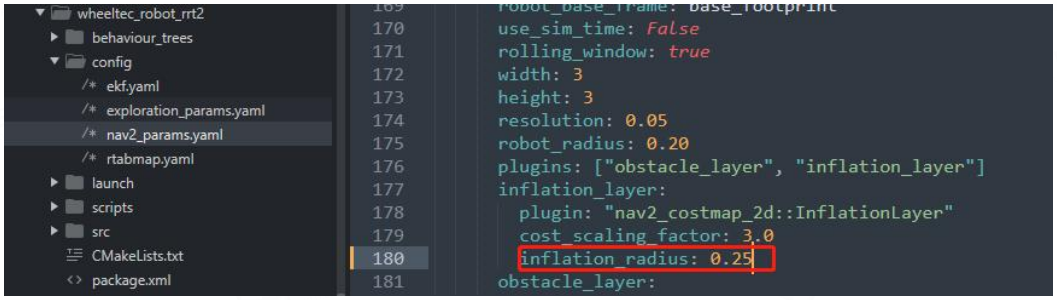

图 7-3-0　nav2_params.yaml 参数文件

```
cd ~/wheeltec_ros2
colcon build --packages-select wheeltec_robot_rrt
```

② 运行 RRT Exploration

在 ROS2 中运行 RRT Exploration 自主探索建图：

先运行 2D 建图：

```
ros2 launch wheeltec_slam_toolbox online_sync.launch.py
```

再打开 RRT 自主探索建图：

```
ros2 launch wheeltec_robot_rrt wheeltec_rrt_slam.launch.py
```

运行效果如图所示：

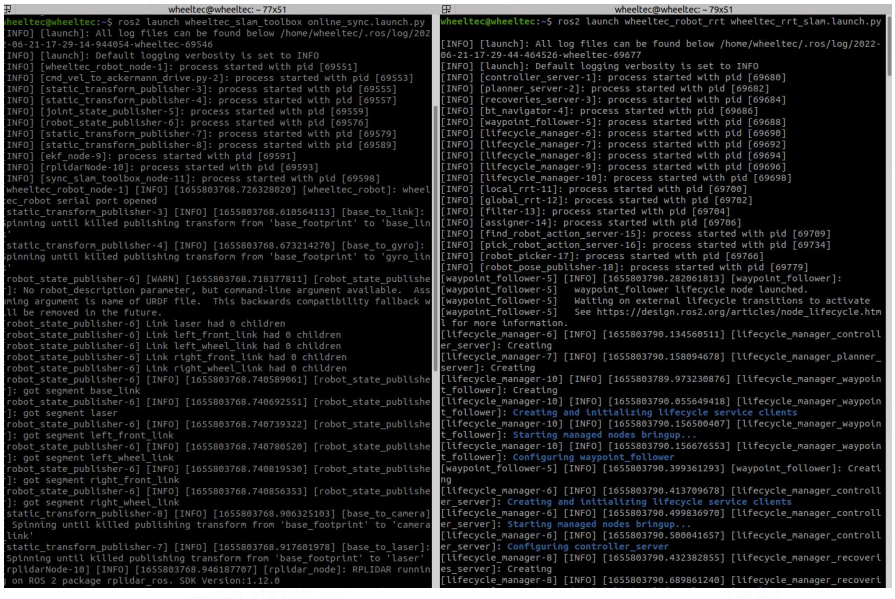

图 7-3-1　终端运行 rrt slam

期间终端会刷一段时间如下信息，这属于正常现象：

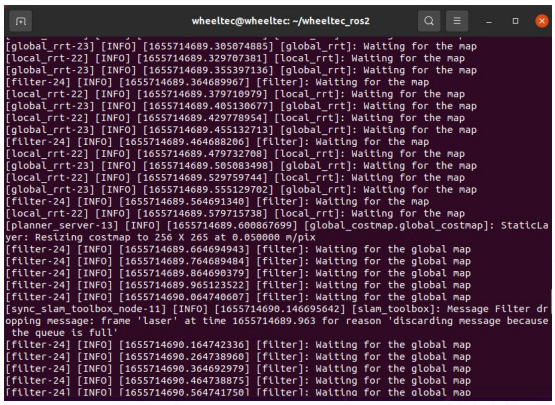

图 7-3-2　终端出现警报

最终终端输入蓝色字体提示后，打开 rviz2

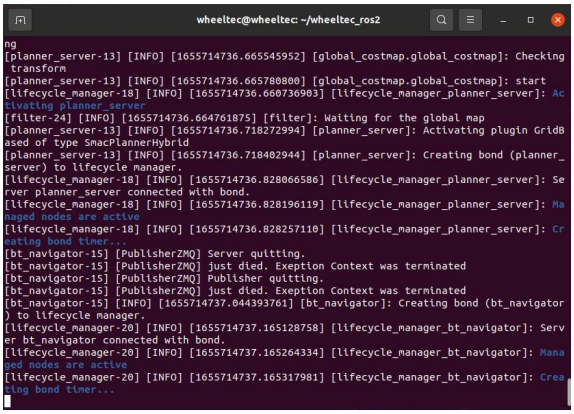

图 7-3-3 终端最终输出

在虚拟机端打开 RVIZ2：

```
ros2 launch wheeltec_rviz2 wheeltec_rviz.launch.py
```

注意：该 rviz 文件是环境中所配置好的，若使用其它 rviz 文件，请添加配置信息：

点击 rviz 界面左下角的 add，在弹窗中选择 By topic，添加图中框选的六个话题。

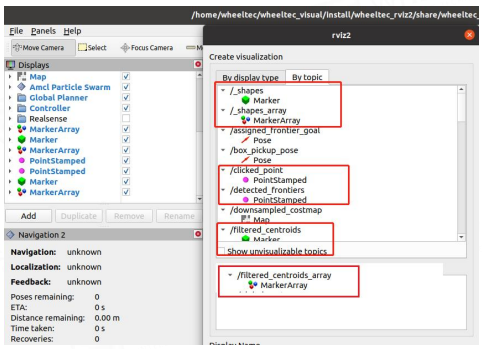

图 7-3-4　rviz2 配置

点击 Publish Point 按顺时针或逆时针的顺序发布建图区域的四个边缘点：

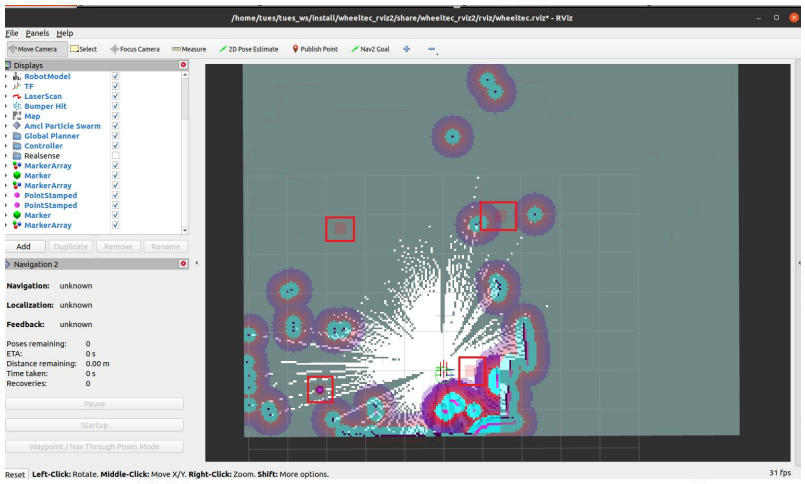

图 7-3-5　发布边缘点

顺时针发布四次Point后，用户还需要定下第五个点（最后一次PublishPoint），这个点是随机树的起点，需要发布在已知地图区域，一般选择定在小车车身上。当雷达扫描范围有部分选择屏蔽时，　需将起点定在距离小车较近处、雷达可扫描范围内。

注意：如果没有按照顺时针或逆时针顺序选取范围点，则所选范围点无法围成闭合图形，无法启动自主建图。

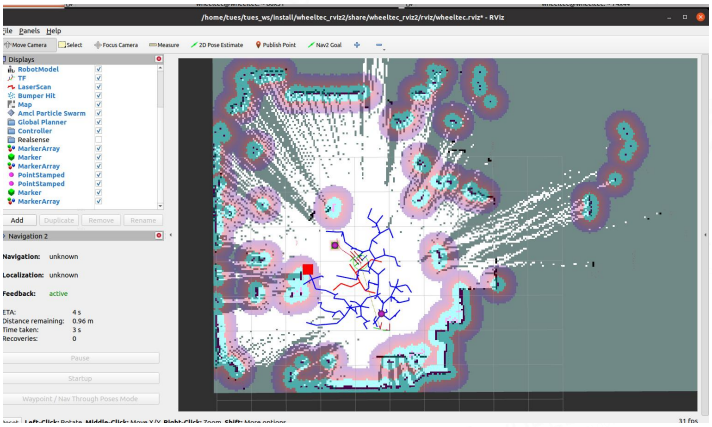

图 7-3-6　开始出现生长树

发布完最后一个点后，随机树开始生长，机器人开始进行导航并对地图边界进行探索。建图过程中蓝色树状为全局树。

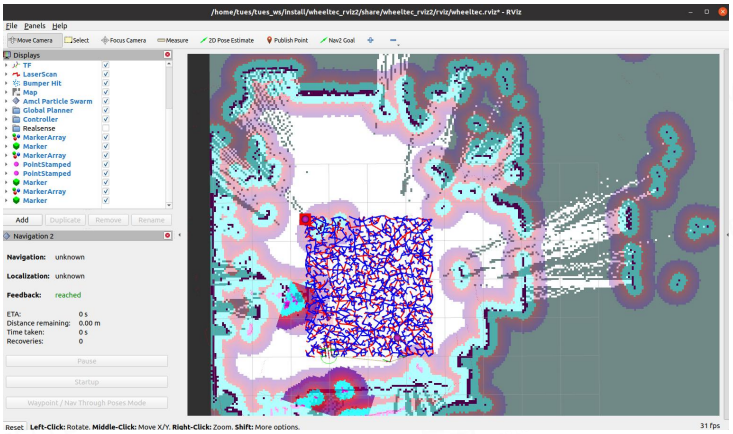

图 7-3-7　密集版生长树

完成探索后，小车将停止运动，此时开始进行计数，若 20 秒内小车都保持静止，则认为自主建图已经完成，启动保存地图节点。

运行 RRT Exploration 时查看 Tf Tree 与节点关系图为：

Tf Tree：

https://gitee.com/tuesdg/wheeltec-ros2-rrt-exploration/blob/master/frames.pdf

节点关系图：

https://gitee.com/tuesdg/wheeltec-ros2-rrt-exploration/raw/master/rosgraph.png

ROS2 中的 RRT Exploration 算法是根据 ROS1 移植的，其原理和 ROS1 一致。

7.4 Wheeltec_robot_rtab[RTAB 建图导航]

wheeltec_robot_rtab 功能包主要实现 WHEELTEC ROS2 的 3D 建图功能。

RTAB-Map (Real-Time 外观- based Mapping)是一种基于全局贝叶斯环路闭合检测器的 RGB-D 图 SLAM 方法，通过视觉 slam 结合激光 slam 的方式来实现视觉建图与激光雷达建图并行。关于 rtabmap 的详细介绍可以查看相应的 ROS wiki 页面：http://wiki.ros.org/rtabmap。

WHEELTEC ROS2 机器人使用 Rtabmap 算法时需要用到相机和雷达，镜像默认使用 Astra S 相机。若用户使用的是 AstraPro 相机，可以查看这一小节修改文件。

在 ROS2 中安装 rtabmap 算法：

```
sudo apt install ros-galactic-rtabmap*
```

① rtabmap 建图

使用 Astra S 相机需要通过 ros1_bridge 传输图像数据。Rtabmap 建图指令为：

```
ros2 launch wheeltec_robot_rtab wheeltec_slam_rtab.launch.py
```

在虚拟机端打开 rviz 可视化查看 3d 建图效果：

运行效果如下：

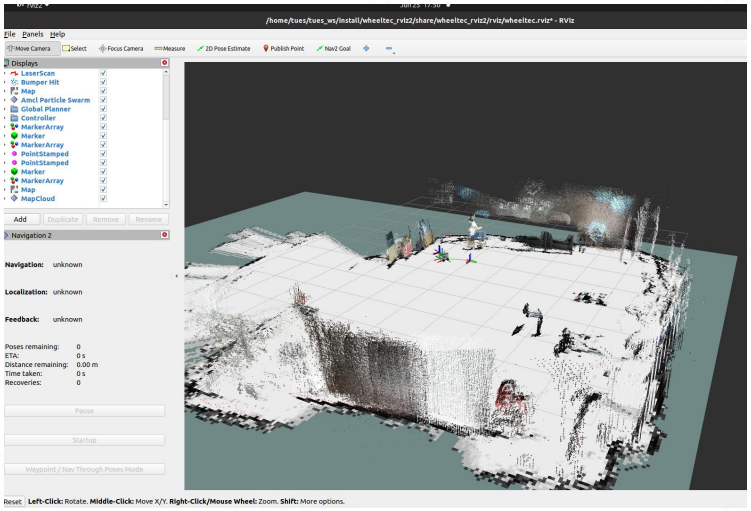

图 7-4-1　ROS2 RTAB-MAP 建图效果

建图完成后，以 ctrl+c 结束进程，rtabmap 会自动保存地图，构建的地图默认存放在~/.ros/rtabmap.db 路径下，如果要查看地图，进入此路径打开终端，使用指令查看：

```
cd ~/.ros/rtabmap.db
rtabmap-databaseViewer rtabmap.db
```

如图所示：

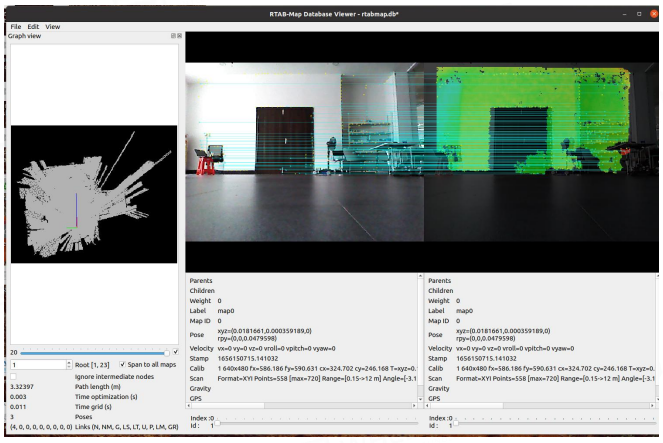

图 7-4-2　使用 RTAB-MAP 工具查看地图

② rtabmap 导航

Rtabmap 算法构建的地图是 db 格式，和 2D 建图的地图格式并不一致。 需

要将 Nav2 链接到 rtab-map 主题并将地图实时提供给 Nav2，实际上 rtab-map 默认已经与 Nav2 集成了，在导航中，需要先启动 RTAB 地图再启动 Nav2。这里演示的 rtabmap 导航，只是使用了 rtab-map 算法，而不是将上一步构建的地图导入到导航中。由于 rviz2 的问题，暂时无法将构建好的 3d 地图加载到 nav2 rviz2 中。

打开 rtab-map 导航指令为：

```
ros2 launch wheeltec_robot_rtab wheeltec_nav2_rtab.launch.py
```

运行效果为：

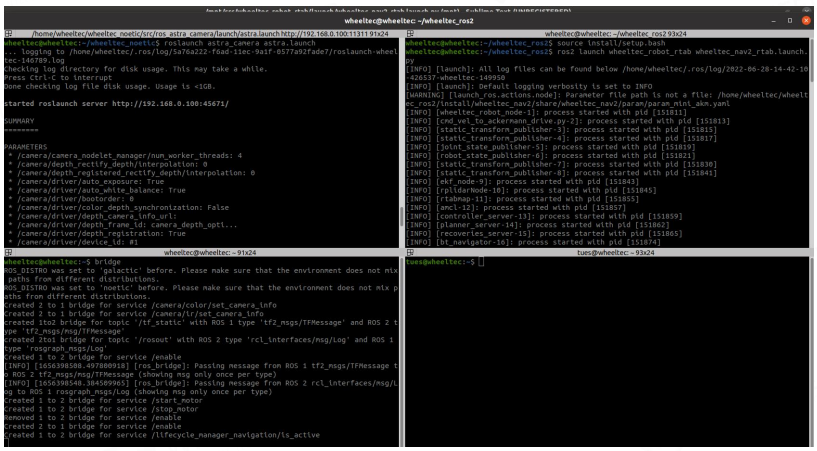

图 7-4-3　输入 RTAB-MAP 导航指令

运行指令后需要等待一段时间，直至终端出现蓝色提示信息：

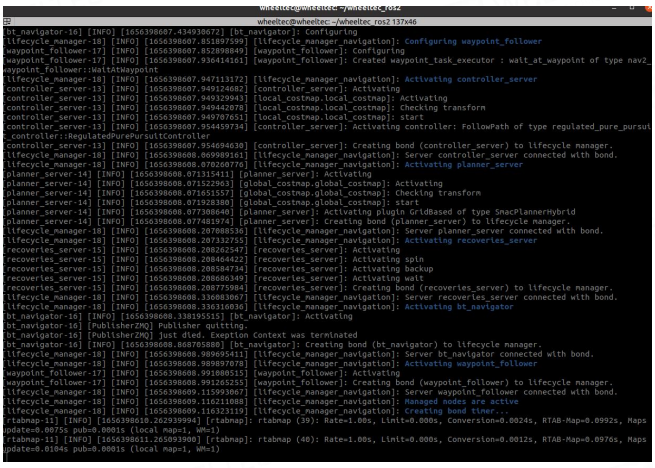

图 7-4-4　终端最终输出

此时打开 rviz2，3d 效果暂未显示，而且 2d 地图没有加载出来，需要先发布目标点，等小车运行一段时间后才会有 3d 地图。打开 rviz2 界面如图所示：

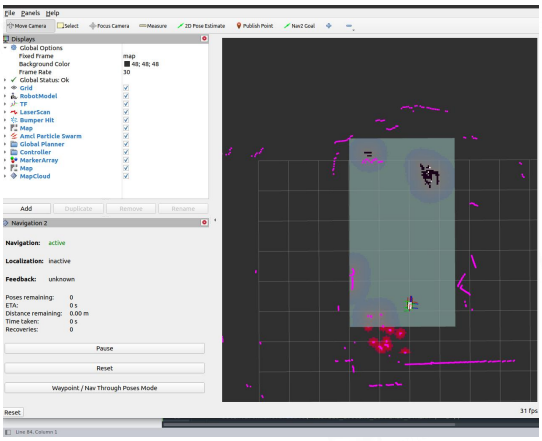

图 7-4-5　rviz2 界面

发布导航点之后即可正常导航，并将 rtabmap 地图可视化：

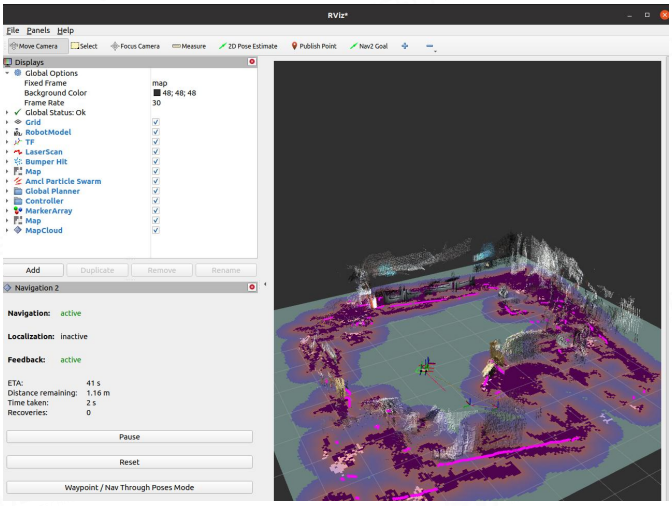

图 7-4-6　rtab-map 导航效果图

7.5 Wheeltec_robot_nav2[2D 导航]

① Nav2 概述

Nav2 项目是 ROS Navigation Stack 的精神继承者。该项目旨在找到一种让移动机器人从 A 点移动到 B 点的安全方法。它也可以应用于涉及机器人导航的其他应用，例如跟随动态点。这将完成动态路径规划、计算电机速度、避开障碍物和结构恢复行为。

Nav2 的预期输入是符合 REP-105 的 TF 转换，如果使用静态成本地图层，则为地图源，BT XML 文件和任何相关的传感器数据源。然后它将为完整或非完整机器人的电机提供有效的速度命令以跟随。

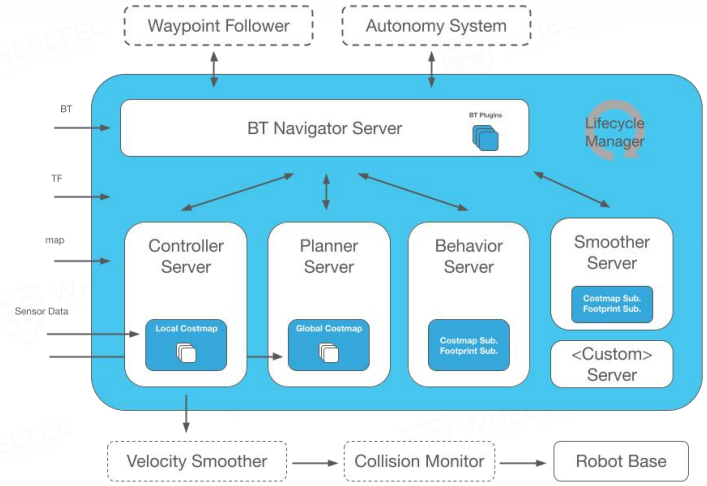

图 7-5-1　Nav2 结构框架图

Nav2 具有以下工具：

- 加载、服务和存储地图（地图服务器）
- 在地图上定位机器人 (AMCL)
- 围绕障碍物规划从 A 到 B 的路径（Nav2 规划器）
- 按照路径控制机器人（Nav2 控制器）
- 平滑路径计划更加连续和可行（Nav2 Smoother）
- 将传感器数据转换为世界的成本地图表示（Nav2 Costmap 2D）
- 使用行为树构建复杂的机器人行为（Nav2 行为树和 BT Navigator）
- 计算发生故障时的恢复行为（Nav2 Recoveries）
- 跟随顺序航点（Nav2 Waypoint Follower）
- 管理服务器的生命周期和看门狗 (Nav2 Lifecycle Manager)
- 启用用户的自定义算法和行为的插件 (Nav2 Core)

NavFn 使用 A* 或 Dijkstra 算法计算从姿势到目标姿势的最短路径。DWB 将使用 DWA 算法来计算遵循路径的控制结果，并使用其自己的几个插件用于轨迹修正。恢复行为包括：等待、旋转、清除成本图和备份。有一套 BT 插件可以调用这些服务器和计算条件。最后，还有一组 Rviz 插件，用于与堆栈交互并控制生命周期。

② ROS1 To ROS2

从 ROS1 到 ROS2，Navigation 的变化是巨大的。而其中最为显著的变化则

是使用行为树来组合各个功能模块，以实现搭乐高积木的开发效果。

ROS1 中的导航主要由 move_base 来组合各个模块的功能。在 move_base 中实现了一个状态机，全局路径规划器和局部路径规划器都是以插件的形式在相应的状态中被调用的。P2P 导航的流程是：先在一个状态里调用全局路径规划器，然后进入到另外一个状态传路径给局部路径规划器，并使其执行该路径。

Nav2 不再是一个单一的整体状态机，而是利用动作服务器和 ROS 2 的低延迟、可靠的通信来分离想法。各个功能的组织则有所不同。全局路径规划功能由各个全局路径规划的插件实现，这些插件被加载到 Planer Server 中。可以通过相应的 action 向 Planer Server 请求全局规划的路径。此时 P2P 导航流程为：行为树上的相应节点会向 Planer Server 请求一条规划到目标点的路径。然后另一个树上的节点与 Controller Server 通信，将规划好的路径发给 Controller Server 去执行。

nav2_bt_navigator 在 move_base 顶层替换为 Action 接口，以使用基于树的操作模型完成导航任务。它使用行为树来实现更复杂的状态机并添加行为作为额外的动作服务器。这些行为树是可配置的 XML。

计划、行为和控制器服务器也是 BT 导航器可以调用来计算的操作服务器。所有 3 个服务器都可以托管许多算法的许多插件，每个插件都可以从导航行为树中单独调用以实现特定行为。提供的默认插件是从 ROS 1 移植而来的，即：DWB、NavFn 以及类似的行为，例如旋转和清除成本图。还添加了等待固定持续时间的新行为。这些服务器通过它们的操作服务器从 BT 导航器调用以计算结果或完成任务。状态由 BT 导航器行为树维护。

所有这些更改使得在启动/运行时用实现相同接口的任何其他算法替换任何这些节点成为可能。

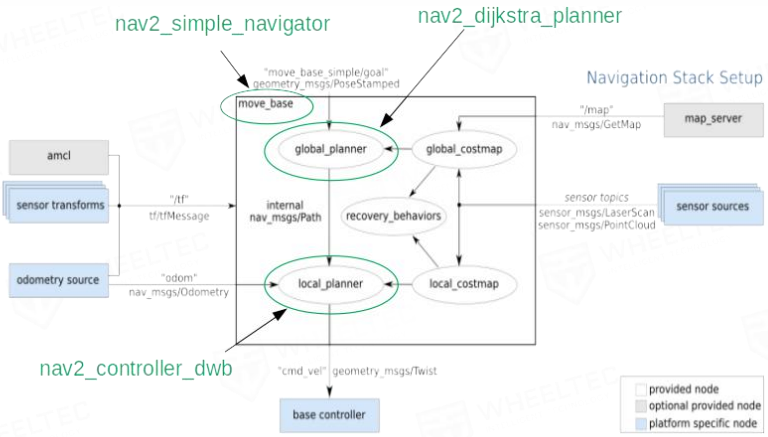

图 7-5-2　Nav2 结构框架图

移植功能包：

amcl：移植到 nav2_amcl

map_server：移植到 nav2_map_server

nav2_planner：替换 global_planner，托管 N 规划器插件

nav2_controller：替换 local_planner，托管 N 控制器插件

Navfn：移植到 nav2_navfn_planner

DWB：替换 DWA 并移植到 nav2_dwb_controller 元包下的 ROS 2

nav_core：移植为 nav2_core 并更新接口

costmap_2d：移植为 nav2_costmap_2d

新功能包：

nav2_bt_navigator：替换 move_base 状态机

nav2_lifecycle_manager：处理服务器程序生命周期

nav2_waypoint_follower：可以通过多个航点来执行复杂的任务

nav2_system_tests: 一组 CI 集成测试和模拟基础教程

nav2_rviz_plugins：一个 rviz 插件，用于控制 Navigation2 服务器、命令、取消和导航

nav2_experimental：深度强化学习控制器的实验（和不完整）工作

navigation2_behavior_trees：行为树库的包装器，用于调用 ROS 动作服务器

③ WHEELTEC NAV2 的路径规划器

WHLLTEC -Nav2 使用 Nav2 Planner 的服务器插件 SmacPlanner 作为规划师，使用 SmacPlannerHybrid 分支，使用 Reeds-shepp 运动模型的 SE2 Hybrid-A* 实现，具有更平滑和多分辨率的查询。支持差速、全向以及阿克曼运动模型。关于 SmacPlanner 的更多信息请查看：

https://github.com/ros-planning/navigation2/tree/main/nav2_smac_planner

局部轨迹规划器使用 RegulatedPurePursuitController 算法计算导航路径，它实现了纯追踪算法的变体来跟踪路径。这种变体我们称之为受监管的纯追踪算法，因为它对碰撞和线速度有额外的监管条款。它还实现了 Adaptive Pure Pursuit 算法背后的基础知识，以根据当前速度改变前瞻距离。这个插件实现了 nav2_core::Controller 接口，允许它在导航堆栈中用作控制器服务器的动作服务器（controller_server）中的本地轨迹规划器。

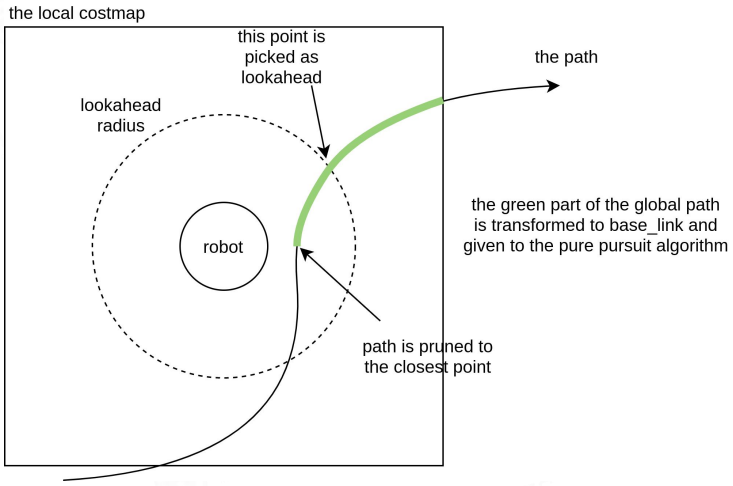

图 7-5-3 本地规划器

该控制器适用于所有类型的机器人，包括差速器、腿式和阿克曼转向车辆。它也可以用于全向平台，但不能充分利用底座的横向运动。如果需要用到横向运动，可以使用 DWB 算法，目前镜像默认使用 RegulatedPurePursuitController 做局部轨迹规划器。

④ WHEELTEC NAV2

wheeltec_nav2 功能包的主要作用是在机器人端实现 2D 导航与其它拓展性

功能。

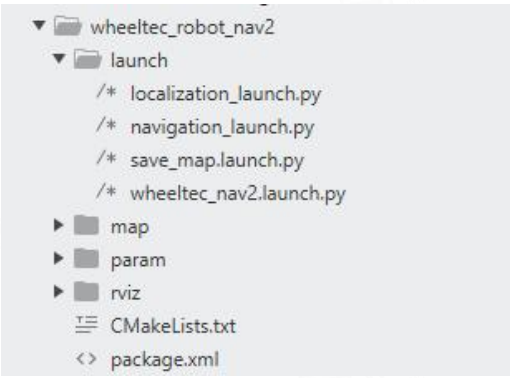

图 7-5-4　wheeltec_nav2 内容

localization_launch.py 是启动机器人在地图中的定位，使用 amcl 蒙特卡洛定位，以及打开 map_server 节点加载地图；

navigation_launch.py 启动导航的一些插件，包括规划器、恢复器和行为树；

在 wheeltec_nav2.launch.py 文件中，除了调用 navigation_launch.py 和 localization_launch.py 文件，还打开了初始化底层控制节点和雷达启动节点，将导航的地图传入 localization_launch.py 中，Nav2 的插件参数和行为树的相关参数传入 navigation_launch.py 文件。在 wheeltec_nav2.launch.py 文件中，如果设置 autostart：= False，则需要单击 RViz 中的开始按钮以初始化节点。同时需要确保 use_sim time 设置为 False，因为我们要使用系统时间而不是 Gazebo 中的模拟时间。

map 文件夹存放导航所需要的地图，所以导航之前需要先构建地图。

param 文件夹存放 Nav2 的插件参数和行为树的 yaml 文件，子文件夹 wheeltec_param 存放 WHLLTEC 不同车型的插件参数。

1) 点到点导航：

在 WHEELTEC 机器人中运行 Navigation2：

```
ros2 launch wheeltec_nav2 wheeltec_nav2.launch.py
```

虚拟机/Ubuntu 电脑中打开 rviz2:

```
ros2 launch wheeltec_rviz2 wheeltec_rviz.launch.py
```

打开 RVIZ2 点击【Nav2 Goal】或【Setgoal】发布导航点如图所示：

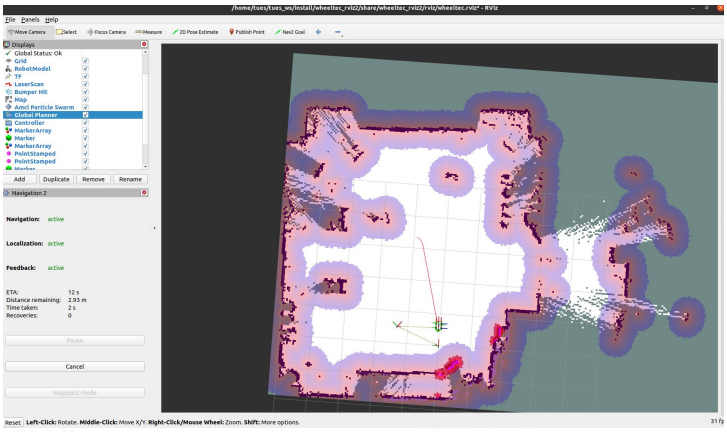

图 7-5-5　运行导航时的 rviz2

在 Nav2 RViz 的左下角方向，用户可以直接通过 RVIZ2 面板找到预计到达时间、距离目标的剩余距离、导航开始后经过的时间以及导航操作期间执行的恢复次数等信息。

2) 路标跟随演示

路标跟踪是导航系统的基本功能。它告诉我们的系统如何使用导航到达多个目的地。在 2D 单点导航的步骤下，打开 rviz2 后，点击左下角的”waypoint mode”切换成路标跟随模式，如图 3-2-7 所示。

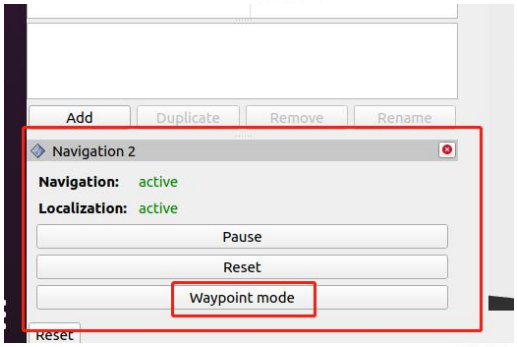

图 7-5-6　waypoint mode 设置页面

到这一步，再点击 rviz2 上方的”Nav2 Goal”开始设定路标位置

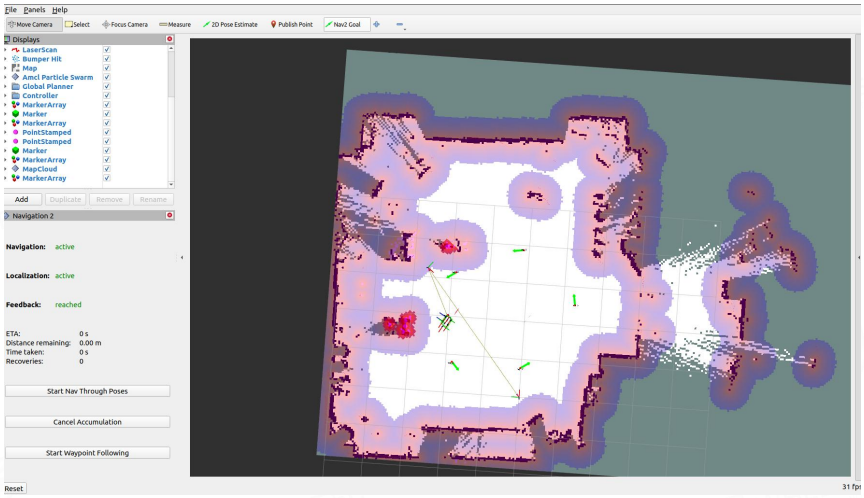

图 7-5-7　waypoint mode 设置路标

标完位置后，点击右下角的”Start Waypoint Following ”开始实现路标跟随。途中可以点击 Cancel 取消前往目标点。

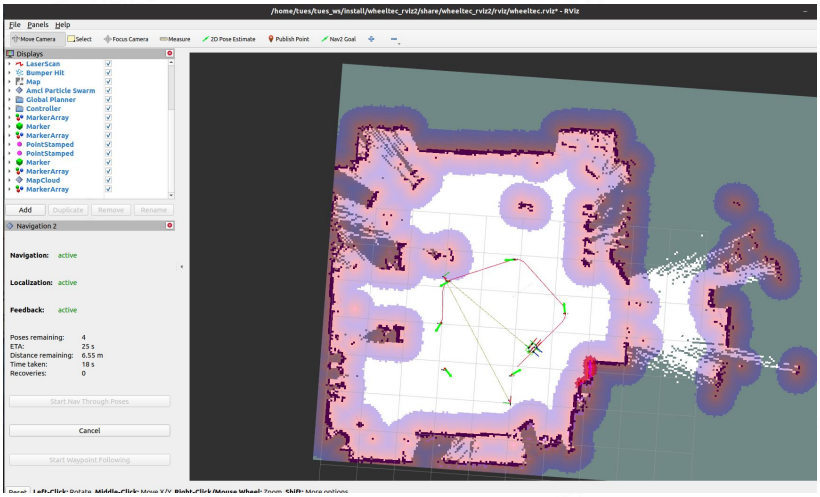

图 7-5-8　waypoint mode 开始导航

7.6Simple_follower[简单跟随控制]

simple_follower_ros2 为简单跟随功能包，这是一个 python 功能包，不含 CPP 文件。启动文件主要有：

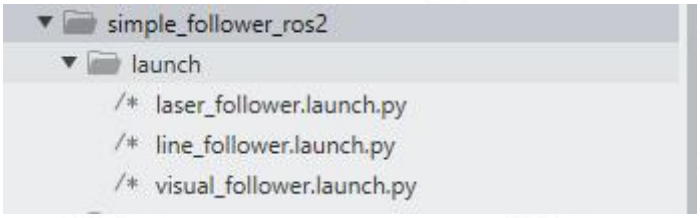

图 7-6-1　simple_follower_ros2 功能包内容

① 雷达跟随

laser_follower.launch.py 启动雷达跟随功能，包括打开底盘控制节点和雷达启动节点。启动命令为：

```
ros2 launch simple_follower_ros2 laser_follower.launch.py
```

雷达跟随启动后，小车会不断寻找雷达扫描范围内距离最近的目标，之后找到一个距离最近且合理的目标进行跟随，最终与被跟随物保持适当的距离（即中距值，预设 0.6m），小车与被跟随物体之间夹角为 0 左右。

注意：开启雷达跟随时尽量保证空间不要过于狭小，物与物之间的距离最好保持 1m 及以上。同时应注意雷达高度，由于雷达只能扫描到与其处于同一水平面上的物体，因此不要将高度较低的物品放置在小车行经路线上，以免发生碰撞。

② 视觉巡线

line_follower.launch.py 启动视觉巡线功能，包括底盘控制节点和 USB 相机启动节点。通过以下指令远程登录到小车端，注意登录时需要添加-Y 参数，开启终端打开弹窗的权限。

```
ssh -Y wheeltec@192.168.0.100
```

运行指令：

```
ros2 launch simple_follower_ros2 line_follower.launch.py
```

运行程序：

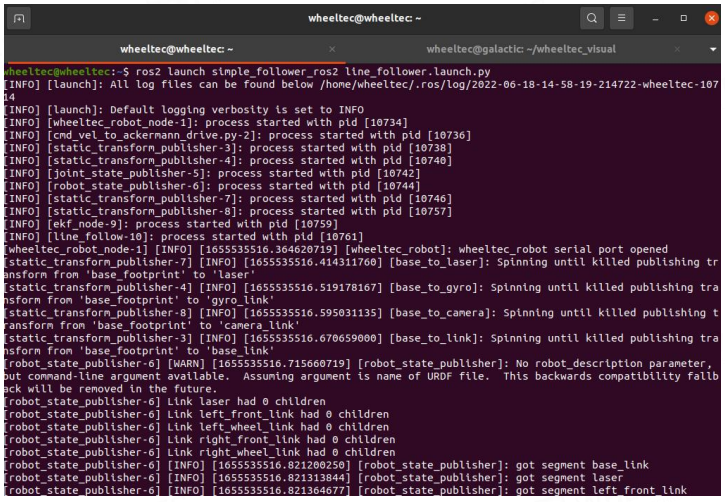

图 7-6-3　运行 line_follower.launch.py

在弹出的弹窗选择小车进行巡线的颜色，以蓝色为例，将滑块滑到【2】，在窗口中白色为识别到指定颜色的结果。

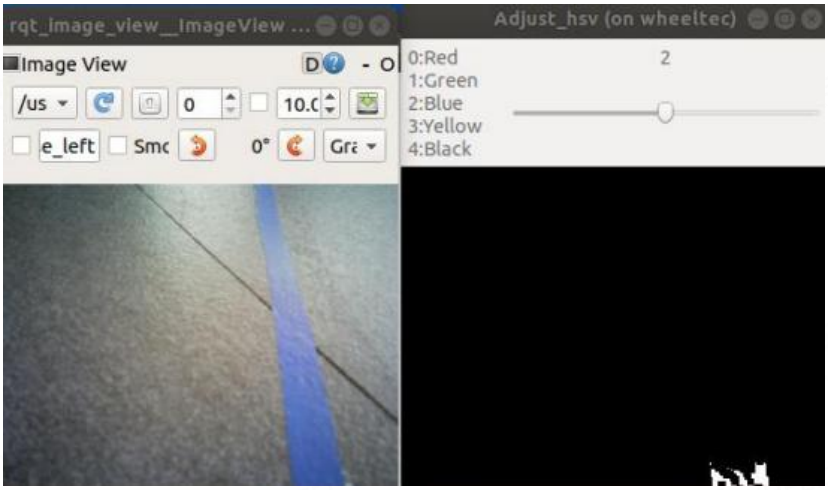

图 7-6-4　运行 line_follower.launch.py 效果

使用 ctrl+c 停止程序后，或许会出现小车依然往前走的现象。这是因为在 rclpy 中暂时没有 ctrl+c 的回调函数用来发布 0 的速度，这个问题将会在下一版本更新解决。此时需要用户手动按下 stm32 的复位按键，小车即可停止运动。

如果想要使用如 C70 之类的 UVC 协议相机进行巡线，可以在巡线启动文件中把参数 bool_usbcam 改为 true，如图 7-6-5 中所示，目前为了适配不同的相机和别名规则，外置了参数接口（默认接口为 C70 相机的串口别名：/dev/Rgbcam），可自行根据需求修改。

```
~/wheeltec_ros2/src/simple_follower_ros2/launch/line_follower.launch.py
```

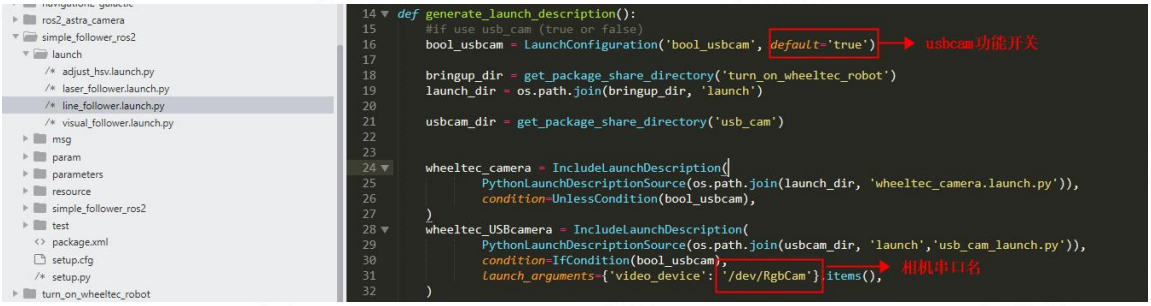

图 7-6-5　line_follower.launch.py 文件

③ 视觉跟踪

visual_follower.launch.py 启动视觉跟随功能，包括底盘控制节点和相机启动节点。默认情况下，机器人视觉跟随目标为**红色物体**，它会通过目标物体的位置进行相关计算获得运动速度，从而实现对目标物体的跟随。

镜像默认使用 Astra S 相机运行视觉巡线功能。运行视觉跟踪功能：

```
ros2 launch simple_follower_ros2 visual_follower.launch.py
```

打开 rqt 工具可以查看色块与当前小车的距离：

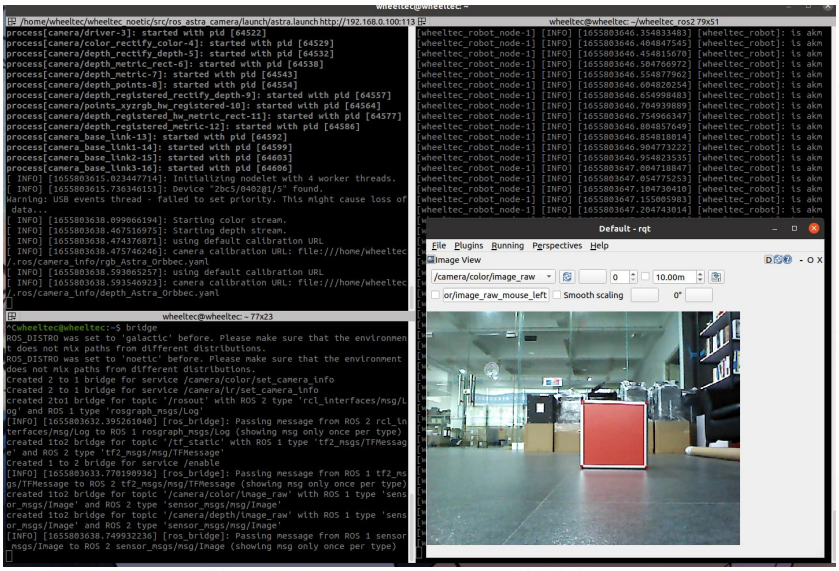

图 7-6-6　运行 visual_follower.launch.py 效果

其中如果使用的是 Gemini 和 Dabai 相机，可以在视觉跟随启动文件中修改 Height 为 400，如图 7-6-7 所示。

```
~/wheeltec_ros2/src/simple_follower_ros2/launch/visual_follower.launch.py
```

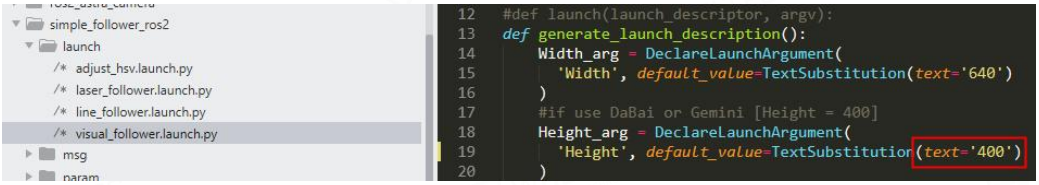

图 7-6-7　visual_follower.launch.py 文件

7.7Wheeltec_web_video[网络视频监控]

wheeltec_web_video 是一个提供网络视频服务器的功能包，它可以实现通过网络进入网页地址查看图像流。

web_video_server 打开一个本地端口并等待传入的 HTTP 请求。一旦通过 HTTP 请求 ROS 图像主题的视频流，它就会订阅相应的主题并创建视频编码器的实例。编码的原始视频数据包提供给客户端。可以通过将附加参数添加到查询字符串来指定参数。

运行网页视频实时监控功能：

Step1：打开相机设备获取图像信息

```
ros2 launch turn_on_wheeltec_robot wheeltec_camera.launch.py
```

Step2：打开 web_video_server 节点，创建等待 HTTP 请求的服务端。

```
ros2 run web_video_server web_video_server
```

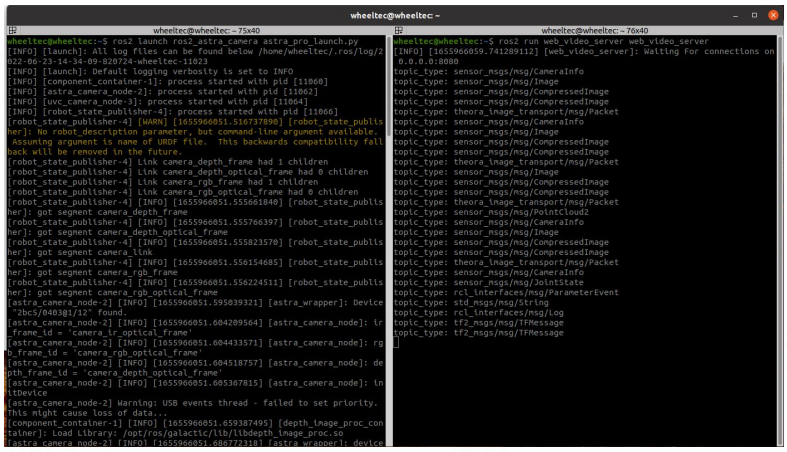

图 7-7-1　打开 web_video_server 节点

Step3：打开客户端（虚拟机）的浏览器并在网页上输入地址 http://192.168.0.100:8080 订阅服务端的图像信息，同时可以通过“ros2 topic list”查看目前已发布的话题列表。

```
主机网页查看：http://localhost:8080/
客户机网页查看：http://192.168.0.100:8080
```

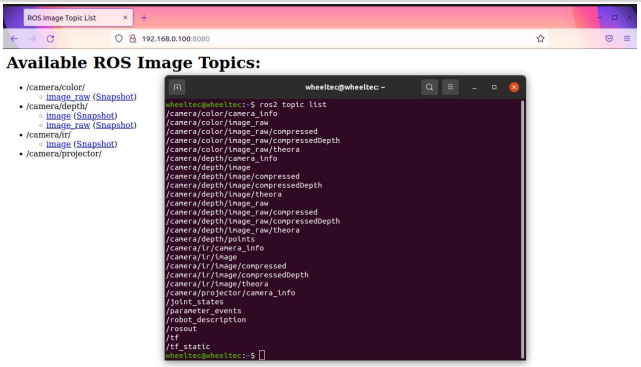

图 7-7-2　在火狐浏览器查看摄像头图像

Step4：在浏览器中点击话题查看相机的实时输出，可以看到 rgb 图与 depth 的图像流，也可以打开 rqt 可视化工具查看相机输出。

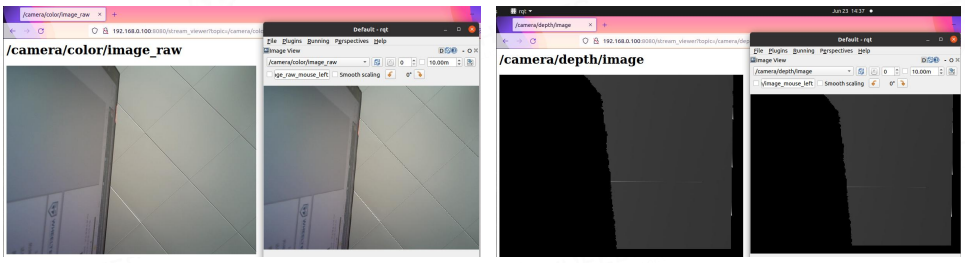

图 7-7-3　查看 RGB 图(左)查看深度图(右)

注意：建议使用谷歌浏览器和火狐浏览器进行图像查看。

7.8Wheeltec_robot_keyboard[键盘控制]

wheeltec_robot_keyboard 实现使用键盘控制操控机器人的功能。运行键盘控制节点时，用户可以通过按下键盘按键发布/cmd_vel 的速度话题。

一般在 WHEELTEC ROS2 机器人打开键盘控制节点前需要先开启底盘控制，使用键盘控制指令为：

```
#开启底盘控制
ros2 launch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch.py
#开启键盘控制
ros2 run wheeltec_robot_keyboard wheeltec_keyboard
```

终端输出如下：

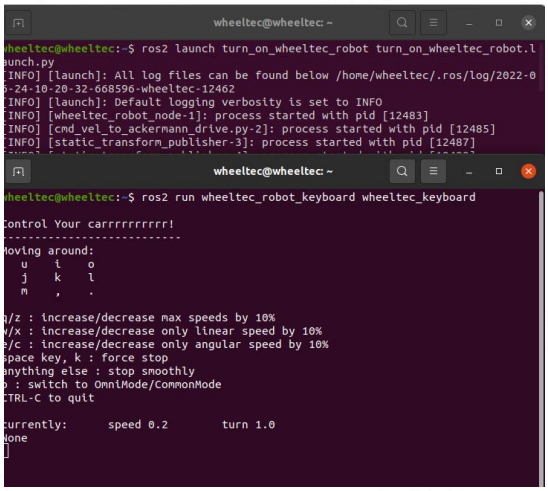

图 7-8-1 wheeltec_keyboard 运行效果

根据终端输出提示，键入[i] [,][j][l]分别代表控制机器人前进后退往左往右，[k]键发布 0 的速度指令使机器人停止运动，符合阿克曼运动模型的机器人无法直接往左或往右运动。

7.9 Wheeltec_robot_joy[Joy 手柄控制]

wheeltec_robot_joy 实现使用 USB 手柄操作机器人的功能。该功能为 ROS 配件附带的功能，需另购 usb 手柄实现。系统中默认已经安装了手柄的驱动 Joy 功能包，该功能包包含 joy_node，这是一个将通用 Linux 游戏杆连接到 ROS 的节点。joy_node 节点会发布“/Joy”话题，其中包含每个操纵杆按钮和轴的当前状态。

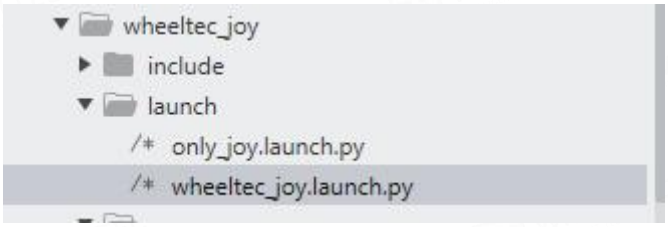

图 7-9-1　wheeltec_robot_joy 功能包内容

only_joy.launch.py 文件只打开手柄启动节点，运行该文件后 wheeltec_joy 节点会发布/cmd_vel 话题数据。

wheeltec_joy.launch.py 文件打开手柄启动节点与底盘控制节点，运行该文件后，wheeltec_joy 节点会发布/cmd_vel 话题数据，底盘控制节点 wheeltec_robot_node 接收/cmd_vel 话题，实现手柄控制机器人运动。

8. 常见问题

本章主要是收集常见问题，大家调试过程中遇到的问题可以反馈给我们，我们会做一个 FQA 文档。

8.1 建图导航使用地图

① 如何在指定目录下保存自定义名称的地图文件？并在导航中使用改地图？

指定路径保存地图指令：

```
ros2 run nav2_map_server map_saver_cli  -f
~/wheeltec_ros2/src/wheeltec_robot_nav2/map/WHEELTEC  --ros-args -p
save_map_timeout:=2000
```

~/wheeltec_ros2/src/wheeltec_robot_nav2/map 为指定保存地图的路径 WHEELTEC 为地图文件名。

在导航中使用指定地图，需要修改文件

```
~/wheeltec_ros2/src/wheeltec_robot_nav2/launch/wheeltec_nav2.launch.py
```

中的 my_map_file 参数，使用绝对路径输入地图名，如图所示：

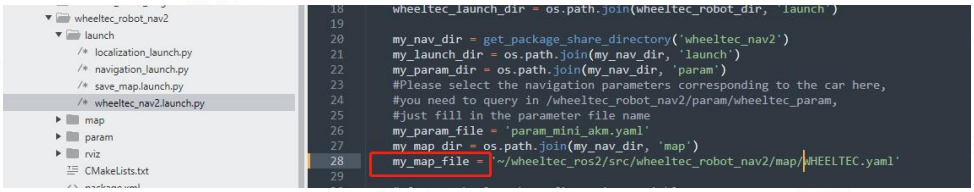

图 8-1-1　修改 wheeltec_nav2.launch.py 内容

修改后进入工作空间路径进行编译即可：

```
cd ~/ wheeltec_ros2
colcon build --packages-select wheeltec_nav2
```

② 如何保存多个地图文件？

使用以下指令保存地图

```
ros2 run nav2_map_server map_saver_cli  -f
~/wheeltec_ros2/src/wheeltec_robot_nav2/map/WHEELTEC
```

将地图文件名 WHEELTEC 改为不同的文件名即可保存多个地图文件。

③ 为何我打开导航时地图是全白的？如何修改成示例中地图的样子？

修改地图文件

```
/wheeltec_ros2/src/wheeltec_robot_nav2/map/WHEELTEC.yaml
```

将参数 free_thresh 的值改为 0.196，如图所示：

图 8-1-2　修改 WHEELTEC.yaml 内容

8.2 常见报错

① [wheeltec_robot_node-1] [ERROR]报错：wheeltec_robot can not open serial port

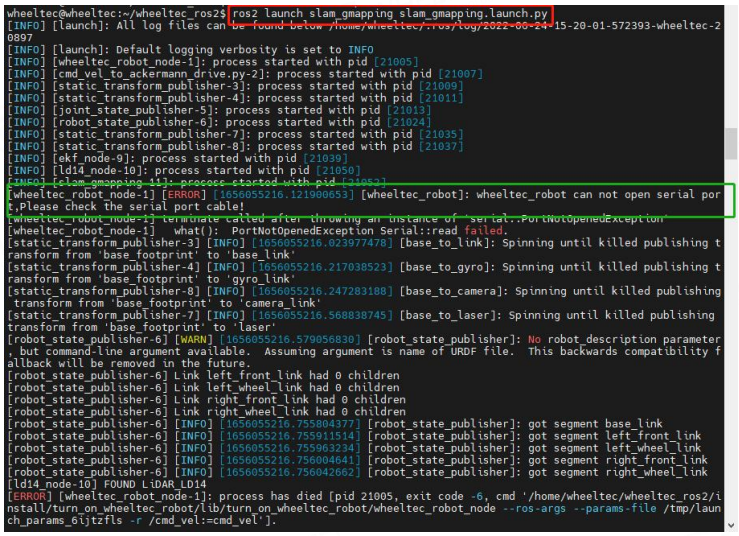

图 8-2-1　问题描述

原因：系统没有识别到 STM32 的串口。

解决：需要确保 stm32 的串口号是 0002，进入目录 /home/wheeltec/wheeltec_ros2/src/turn_on_wheeltec_robot

运行指令：

```
sudo sh wheeltec_udev.sh
```

运行完指令后，拔插串口，使用 ll /dev 查看是否出现

```
wheeltec_controller -> ttyUSB0
```

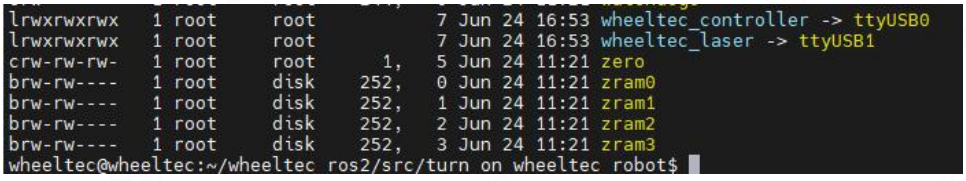

图 8-2-2　设备显示

② ekf node 警报：Invalid frame ID "gyro_link" passed to canTransform argument source_frame - frame does not exist

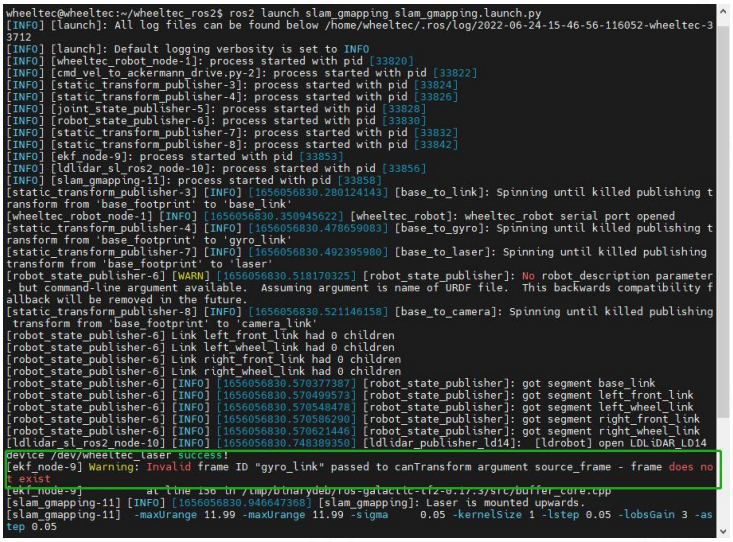

图 8-2-3　问题描述

原因：这是因为关于"gyro_link"的转换没有被广播，或者需要该转换的传感器数据比最近发布的转换更新。不必担心，该警报不影响功能使用。