

轮趣科技

ROS2 入门教程

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	2021/05/02	第一次发布

网址: www.wheeltec.net

序言

本文档主要讲解了 ROS2 系统的使用方法，文档分为四个部分，包括：ROS2 概述、系统环境搭建、ROS2 中命令行工具介绍使用与客户端库的使用。本文档目的是让用户能快速上手使用 ROS2 系统。

注意：本文档是基于 wheeltec_robot 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 ROS1 与 ROS2 共存的虚拟机环境。关于 rviz2 的配置较为繁琐，若用户自行在虚拟机端配置了 ROS2 的开发环境，则请用户另外找我们的工作人员提供 rviz2 的配置文件替换即可。

目录

序言.....	2
1. ROS2 概述.....	4
1.1 ROS2 支持的操作系统.....	5
1.2 ROS2 新特性.....	5
1.3 关于 DDS.....	7
2. 环境搭建.....	9
2.1 系统准备.....	9
2.2 运行测试例程.....	11
3. ROS2 中的常用命令.....	12
3.1 工作空间与功能包的创建.....	12
3.2 常用命令行工具.....	14
3.3 在 wheeltec_robot 中使用 ROS2.....	21
3.4 launch 启动文件说明.....	27
4. 由 ROS1 过度到 ROS2.....	29
4.1 ROS1 与 ROS2 之间的差异.....	29
4.2 ROS1 与 ROS2 的通信.....	32

1. ROS2 概述

ROS (Robot Operating System) 是开放源代码 (OSS) 的机器人应用程序架构。其设计适合构建高性能机器人控制所要求的分布式系统，可以灵活且轻易地变更系统构成。ROS 以 UNIX 系列操作系统的 Ubuntu 作为标准环境，被广泛应用到全球机器人的研究开发领域。随着机器人技术的不断发展，ROS 在各个领域得到了极大的推广和应用。ROS 2 的开发过程继承了 ROS 的主要理念，同时加入了实现商用化和实用化所需要的理念，如：包括嵌入式设备在内的多平台化、实时控制、提高容错性、应对多个机器人等。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

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 语言编写的独立库，用来连接到 ROS2 的客户端库，客户端库主要包括 rclcpp、rclpy 和 rcljava 等。

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。

图 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
Foxy	Ubuntu20.04; macOS 10.14(s, server); Windows10-需配置 Visual Studio 2019
Eloquent	Ubuntu18.04(, on, c-, r 64 和 64) , Ubuntu18.04(, on, c- 32) ; macOS 10.12(s, server); Windows10-需配置 Visual Studio 2019
Dashing	Ubuntu18.04(, on, c- 64 和 64) , Ubuntu18.04(, on, c- 32) ; macOS 10.12(s, server); Windows10-需配置 Visual Studio 2019
Crystal	Ubuntu18.04(, on, c); Ubuntu 16.04(Xenial)-源码安装; macOS 10.12(s, server); Windows10
Bouncy	Ubuntu18.04(, on, c); Ubuntu 16.04(Xenial)-源码安装; macOS 10.12(s, server); Windows10-需配置 Visual Studio 2017
Ardent	Ubuntu 16.04(Xenial)、macOS 10.12(s, server)、Windows10

由于我们的产品使用的环境是 Ubuntu18.04，所以本次适配的 ROS2 版本为 Eloquent，即第五个发行版本。

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 1-1-1 所示。

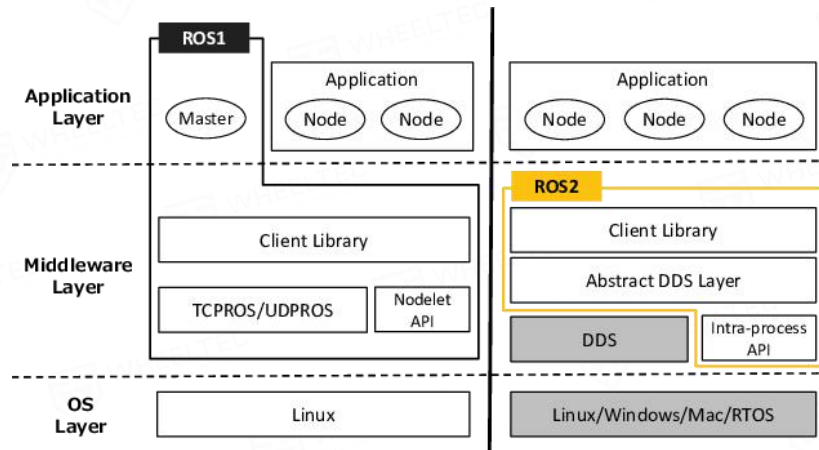


图 1-1-1 ROS1 与 ROS2 架构图

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。

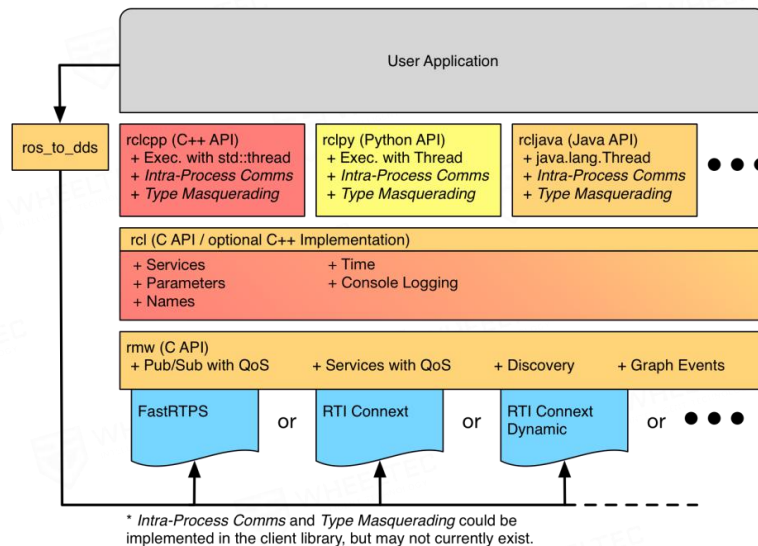


图 1-1-2 ROS2 的 API 架构图

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, 使这些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 通信, 二者共存。
- 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。

1.3 关于 DDS

ROS2 遵循行业标准，通过一个成为 DDS 实现的概念来实现实时通信，在 ROS2 中，操作系统层与底层硬件的通信时通过 DDS 实现完成的，供应商不同则具体实现不同。抽象 DDS 层与 ROS2 客户端库连接，并通过 DDS 实现帮助用户连接代码。所有的 DDS 实现都是通过一个称为 ROS 中间层(RMW)的特殊层实现的。

DDS(数据分发服务)是数据分发服务，DDS 是 OMG 定义的标准。它是一种发布-订阅传输技术。与 ROS1 不同，DDS 实现了一种分布式发现系统技术，可以帮助多个 DDS 程序在不使用 master 的情况下相互通信。DDS 的技术核心是以数据为中心的发布-订阅模型(Data-Center Publish-Subscribe, DCPS)。

在 DDS 中，每个发布者或订阅者都成为参与者，类似于 ROS 中的节点概念。每个参与者都可以使用某个定义的数据类型来读写全局数据空间。它类似于 ROS1 的发布和订阅，那么让我们来比较一下这两种数据发布与订阅通信模型。

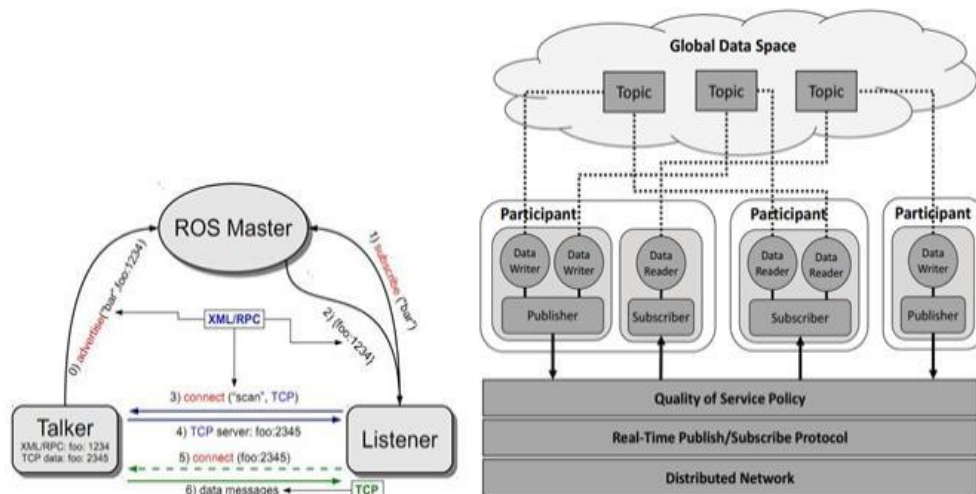


图 1-1-3 ROS1 和 ROS2 的数据发布与订阅

在 ROS1 中，如果一个节点想要发布消息，它需要在主节点中注册，让主节点知道该节点要做什么。如果另一个节点想要订阅消息，它需要询问主节点消息在哪里。DDS 将分布式网络中传输的数据定义为主题 (Topic)，将数据的发布和接收对象分别定义为发布者 (Publisher) 和订阅者 (Subscriber)，从而构成数据的发布/订阅传输模型。各个节点在逻辑上无主从关系，点与点之间都是对等关系，通信方式可以是点对点、点对多、多对多等，在 QoS 的控制下建立

连接，自动发现和配置网络参数。

ROS2 中的 QoS 是网络中跨节点性能的度量。QoS 是 DDS 的一个非常重要的组成部分，它控制着各个方面与底层的通信机制，主要从时限、可靠性、连续性、历史记录等方面进行控制，以满足不同场景下用户的数据应用需求。

2. 环境搭建

2.1 系统准备

① Ubuntu18.04 安装 ROS2-Eloquent

在搭建 ROS2 环境之前首先需要确认系统以及将 ROS1 安装成功。

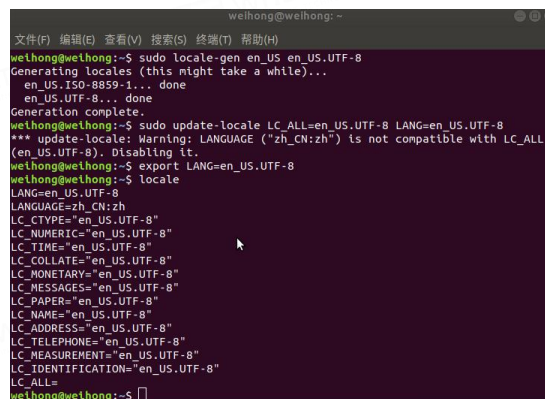
有两种方法安装 ROS2，一种是源码安装，另一种是二进制安装。本教程使用的是二进制安装。源码安装教程详细可以访问[官网教程](#)来了解 ROS2 安装的更多信息。

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，使用一下指令设置：

```
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：



```
welhong@welhong: ~
welhong@welhong:~$ sudo locale-gen en_US en_US.UTF-8
Generating locales (this might take a while)...
  en_US.ISO-8859-1... done
  en_US.UTF-8... done
Generation complete.
welhong@welhong:~$ sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
*** update-locale: Warning: LANGUAGE ("zh_CN:zh") is not compatible with LC_ALL
(en_US.UTF-8). Disabling it.
welhong@welhong:~$ export LANG=en_US.UTF-8
welhong@welhong:~$ locale
LANG=en_US.UTF-8
LANGUAGE=zh_CN:zh
LC_CTYPE=en_US.UTF-8
LC_NUMERIC=en_US.UTF-8
LC_TIME=en_US.UTF-8
LC_COLLATE=en_US.UTF-8
LC_MONETARY=en_US.UTF-8
LC_MESSAGES=en_US.UTF-8
LC_PAPER=en_US.UTF-8
LC_NAME=en_US.UTF-8
LC_ADDRESS=en_US.UTF-8
LC_TELEPHONE=en_US.UTF-8
LC_MEASUREMENT=en_US.UTF-8
LC_IDENTIFICATION=en_US.UTF-8
LC_ALL=
welhong@welhong:~$
```

图 2-1-1 设置完系统区域的终端输出信息

- 2) 添加 ROS2 的代码仓库

使用一下指令授权密钥：

```
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
apt-key add -
```

将代码仓库添加到源列表：

```
sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)]
http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" >
```

```
/etc/apt/sources.list.d/ros2-latest.list'
```

执行完后终端显示如图 2-1-2 所示。

```
weihong@weihong:~$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
OK
weihong@weihong:~$ sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" > /etc/apt/sources.list.d/ros2-latest.list'
weihong@weihong:~$
```

图 2-1-2 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-eloquent-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-pip python-rosdep python3-vcstool wget
```

使用 pip3 安装测试功能包, 执行以下指令:

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

安装 FAST-RTPS 依赖项, 执行以下指令:

```
sudo apt install --no-install-recommends -y libasio-dev libtinyxml2-dev
```

安装 Cyclone DDS 依赖项, 执行以下指令:

```
sudo apt install --no-install-recommends -y libcunit1-dev
```

③ ROS2 的环境设置

到现在为止, 我们使用的 Ubuntu 系统同时安装了 ROS1 和 ROS2, 所以需要进行 ROS1 与 ROS2 共存环境的设置。此时可以打开一个终端运行 `roscore` 命令, 查看 ROS1 的相关设置是否正常: 如图所示。

为了不让系统的环境配置文件搞混两个环境, 用户可以在使用相应的 ROS 版本功能包时, 执行相应的 `source` 命令来导入对应的 ROS 版本的环境。ROS 2

依赖于使用 shell 环境组合工作空间的概念。

为了避免用户每次打开终端都执行一次 source 命令,可以将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,修改完后文件如图 2-1-3 所示

```
alias initros1=" source /opt/ros/melodic/setup.bash"
alias initros2=" source /opt/ros/eloquent/setup.bash"
```

```
# enable programmable completion features (you don't need to enable
# 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

alias initros2="source /opt/ros/eloquent/setup.bash"
alias initros1="source /opt/ros/melodic/setup.bash"
#source /opt/ros/melodic/setup.bash
```

图 2-1-3 修改后的环境配置文件

保存退出脚本文件。使用以下命令使修改生效:

```
source ~/.bashrc
```

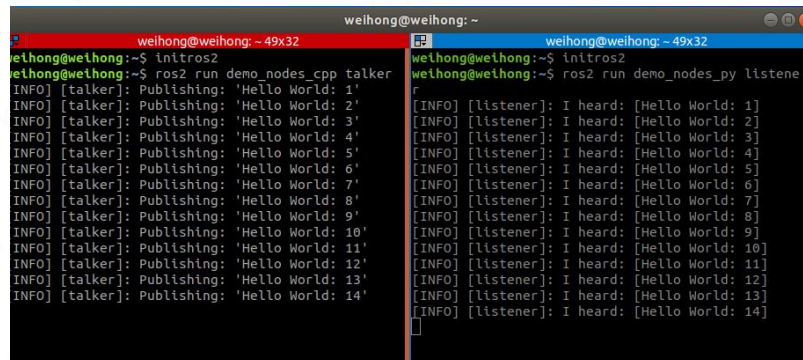
这个修改是为了方便用户可以根据自己的需要,使用"initros1"或"initros2"命令调用系统环境。

2.2 运行测试例程

- 1) 使用 ctrl+alt+t 打开一个终端,输入命令"initros2"初始化 ROS2 环境;
- 2) 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:



```
weihong@weihong: ~ - 49x32
weihong@weihong:~$ initros2
weihong@weihong:~$ ros2 run demo_nodes_cpp talker
INFO [talker]: Publishing: 'Hello World: 1'
INFO [talker]: Publishing: 'Hello World: 2'
INFO [talker]: Publishing: 'Hello World: 3'
INFO [talker]: Publishing: 'Hello World: 4'
INFO [talker]: Publishing: 'Hello World: 5'
INFO [talker]: Publishing: 'Hello World: 6'
INFO [talker]: Publishing: 'Hello World: 7'
INFO [talker]: Publishing: 'Hello World: 8'
INFO [talker]: Publishing: 'Hello World: 9'
INFO [talker]: Publishing: 'Hello World: 10'
INFO [talker]: Publishing: 'Hello World: 11'
INFO [talker]: Publishing: 'Hello World: 12'
INFO [talker]: Publishing: 'Hello World: 13'
INFO [talker]: Publishing: 'Hello World: 14'

weihong@weihong: ~ - 49x32
weihong@weihong:~$ initros2
weihong@weihong:~$ ros2 run demo_nodes_py listener
INFO [listener]: I heard: [Hello World: 1]
INFO [listener]: I heard: [Hello World: 2]
INFO [listener]: I heard: [Hello World: 3]
INFO [listener]: I heard: [Hello World: 4]
INFO [listener]: I heard: [Hello World: 5]
INFO [listener]: I heard: [Hello World: 6]
INFO [listener]: I heard: [Hello World: 7]
INFO [listener]: I heard: [Hello World: 8]
INFO [listener]: I heard: [Hello World: 9]
INFO [listener]: I heard: [Hello World: 10]
INFO [listener]: I heard: [Hello World: 11]
INFO [listener]: I heard: [Hello World: 12]
INFO [listener]: I heard: [Hello World: 13]
INFO [listener]: I heard: [Hello World: 14]
```

图 2-2-1 ROS2 中的发布者(左)与订阅者(右)的输出

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) 输入命令”initros2”调出 ROS2 环境
- 2) 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:

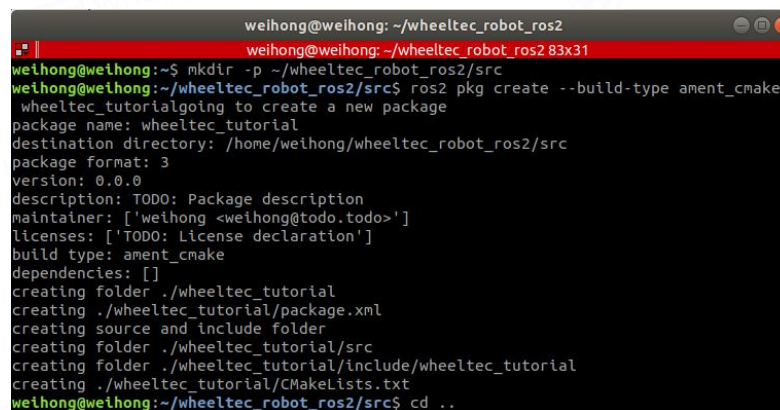
- 1) 输入命令”initros2”调出 ROS2 环境
- 2) 进入工作空间的 src 文件夹: cd ~/wheeltec_robot_ros2/src
- 3) 创建一个名为 wheeltec_tutorials 的功能包:

```
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。



```
weihong@weihong: ~/wheeltec_robot_ros2  
weihong@weihong: ~/wheeltec_robot_ros2 83x31  
weihong@weihong:~$ mkdir -p ~/wheeltec_robot_ros2/src  
weihong@weihong:~/wheeltec_robot_ros2/src$ ros2 pkg create --build-type ament_cmake  
wheeltec_tutorialgoing to create a new package  
package name: wheeltec_tutorial  
destination directory: /home/weihong/wheeltec_robot_ros2/src  
package format: 3  
version: 0.0.0  
description: TODO: Package description  
maintainer: ['weihong <weihong@todo.todo>']  
licenses: ['TODO: License declaration']  
build type: ament_cmake  
dependencies: []  
creating folder ./wheeltec_tutorial  
creating ./wheeltec_tutorial/package.xml  
creating source and include folder  
creating folder ./wheeltec_tutorial/src  
creating folder ./wheeltec_tutorial/include/wheeltec_tutorial  
creating ./wheeltec_tutorial/CMakeLists.txt  
weihong@weihong:~/wheeltec_robot_ros2/src$ cd ..
```

图 3-1-1 ROS2 创建一个功能包

4) 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
```

5) “工作区”是一个 ROS 术语，表示用户在系统中使用 ROS 2 开发的位置。在使用 ROS 2 开发时，通常会同时激活多个工作区。工作区通过 `setup.bash` 进行设置和切换。返回到 `wheeltec_robot_ros2` 目录下，运行以下命令以获取工作区：

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

6) 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 3-1-2 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 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` 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图 3-1-3 所示。

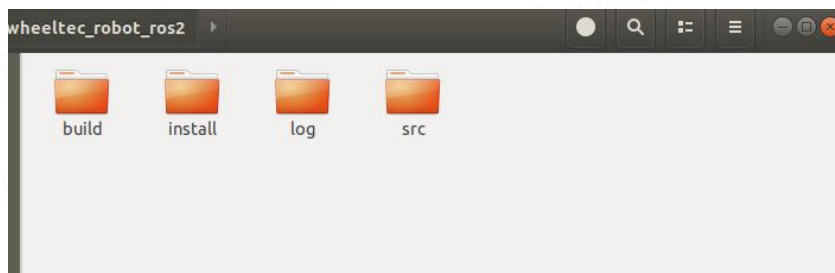


图 3-1-3 工作空间文件夹

`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。

3.2 常用命令行工具

ROS2 中的命令行工具与 ROS1 中的命令行工具使用方法相差不大，本章只使用 ROS2 节点与 ROS2 话题做详细举例说明；如果用户要为自己的项目编写特定的功能包，推荐完整学习 ROS2 中的[用户手册](#)，其中包含更加全面的指导信息。

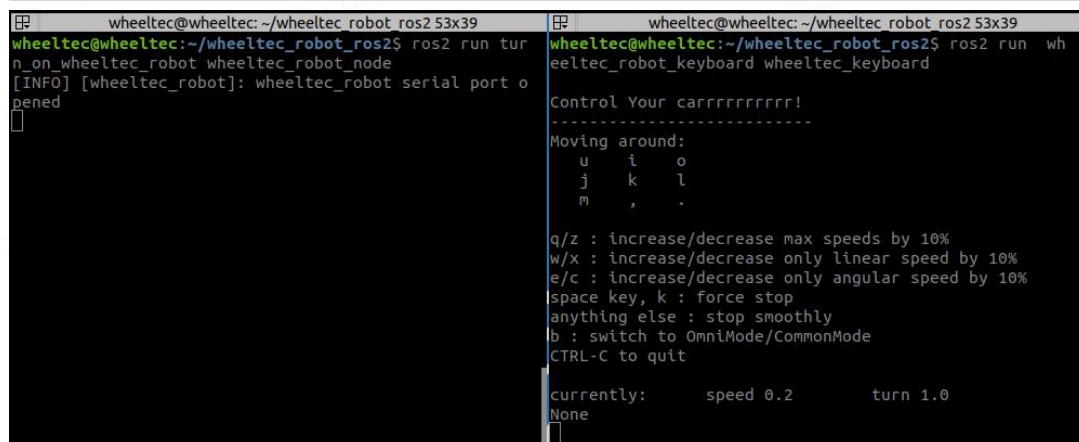
① ROS2 节点

节点是基本的 ROS 2 元素，可在机器人系统中用于单个模块化目的。每个节点都可以通过主题，服务，操作或参数向其他节点发送和接收数据。完整的机器人系统由许多协同工作的节点组成。在 ROS 2 中，单个可执行文件可包含一个或多个节点。以下以打开 wheeltec_robot 节点与键盘控制节点为示例，简单说明节点相关指令的使用方式。注：执行指令前终端需要先调用 ROS2 的系统环境。

```
initros2
cd ~/wheeltec_robot_ros2
sh install/setup.bash
```

1) ROS2 运行可执行文件指令格式：

```
ros2 run <package_name> <executable_name>
#如：打开 wheeltec_robot 节点
ros2 run turn_on_wheeltec_robot wheeltec_robot_node
#程序包名称为 turn_on_wheeltec_robot，可执行文件名称为 wheeltec_robot_node
#打开键盘控制节点
ros2 run wheeltec_robot_keyboard wheeltec_keyboard
```



The figure shows two terminal windows. The left window shows the execution of ROS2 commands to start the robot node and the keyboard control node. The right window shows the output of the keyboard control node, displaying a control interface with movement instructions and current speed/turn values.

```
wheeltec@wheeltec: ~/wheeltec_robot ros2 53x39
wheeltec@wheeltec:~/wheeltec_robot_ros2$ ros2 run turn_on_wheeltec_robot wheeltec_robot_node
[INFO] [wheeltec_robot]: wheeltec_robot serial port opened
wheeltec@wheeltec:~/wheeltec_robot ros2 53x39
wheeltec@wheeltec:~/wheeltec_robot_ros2$ ros2 run wheeltec_robot_keyboard wheeltec_keyboard
Control Your carrrrrrrrrrrr!
-----
Moving around:
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
b : switch to OmniMode/CommonMode
CTRL-C to quit

currently:      speed 0.2      turn 1.0
None
```

图 3-2-1 打开 wheeltec_robot 节点与键盘控制节点

- 2) 使用 `ros2 node list` 指令展示所有正在运行的节点名称:

```
wheeltec@wheeltec:~$ ros2 node list
/wheeltec_keyboard
/wheeltec_robot
```

图 3-2-2 节点列表

- 3) 使用 `ros2 node info` 查询 `wheeltec_robot` 节点的详细信息；在终端中返回与该节点交互的订阅者、发布者、服务和操作（ROS 图连接）的列表。输出应图 3-2-3 所示:

```
wheeltec@wheeltec:~$ ros2 node info /wheeltec_robot
/wheeltec_robot
Subscribers:
  /ackermann_cmd: ackermann_msgs/msg/AckermannDriveStamped
  /akn_cmd_vel: geometry_msgs/msg/Twist
  /parameter_events: rcl_interfaces/msg/ParameterEvent
Publishers:
  /PowerVoltage: std_msgs/msg/Float32
  /mobile_base/sensors/imu_data: sensor_msgs/msg/Imu
  /odom_combined: nav_msgs/msg/Odometry
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /robotpose: wheeltec_robot_msgs/msg/Data
  /robotvel: wheeltec_robot_msgs/msg/Data
  /rosout: rcl_interfaces/msg/Log
  /tf: tf2_msgs/msg/TFMessage
Service Servers:
  /wheeltec_robot/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /wheeltec_robot/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /wheeltec_robot/get_parameters: rcl_interfaces/srv/GetParameters
  /wheeltec_robot/list_parameters: rcl_interfaces/srv/ListParameters
  /wheeltec_robot/set_parameters: rcl_interfaces/srv/SetParameters
  /wheeltec_robot/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
Service Clients:
Action Servers:
Action Clients:
```

图 3-2-3 wheeltec 节点详细信息

- 4) 重新映射允许用户将默认节点属性（例如节点名称，主题名称，服务名称等）重新分配给自定义值。输入以下指令将 `wheeltec_keyboard` 节点重命名为 `new_key`

```
ros2 run wheeltec_robot_keyboard wheeltec_keyboard --ros-args --remap
__node:=new_key
```

```
wheeltec@wheeltec:~/wheeltec_robot_ros2$ ros2 run wh
wheeltec_robot_keyboard wheeltec_keyboard --ros-args --
remap __node:=new_key

Control Your carrrrrrrrrrr!
-----
Moving around:
  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
b : switch to OmniMode/CommonMode
CTRL-C to quit

currently:      speed 0.2      turn 1.0
None
█
```

图 3-2-4 wheeltec_keyboard 节点重命名(1)

重映射后节点列表:

```
wheeltec@wheeltec:~$ ros2 node list
/new_key
/wheeltec_robot
wheeltec@wheeltec:~$
```

图 3-2-5 wheeltec_keyboard 节点重命名(2)

5) 终端输入以下指令查看节点之间的可视化信息:

```
initros2
rqt_graph
```

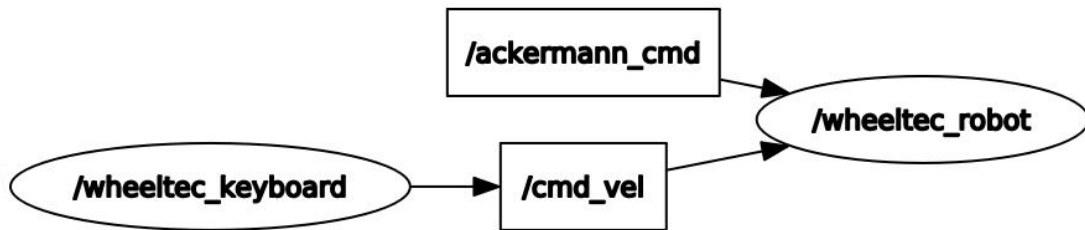


图 3-2-5 节点之间的可视化信息

从图中可以看到，目前正在运行的节点是 wheeltec_keyboard 与底盘控制 wheeltec_robot 节点。

② ROS2 话题

ROS 2 将复杂的系统分解为许多模块化节点。话题是 ROS 图的重要元素，它充当节点交换消息的总线。节点可以将数据发布到任意数量的话题，并同时具有对任意数量的话题的订阅。话题是数据在节点之间以及因此在系统的不同部分之间移动的重要方式之一。

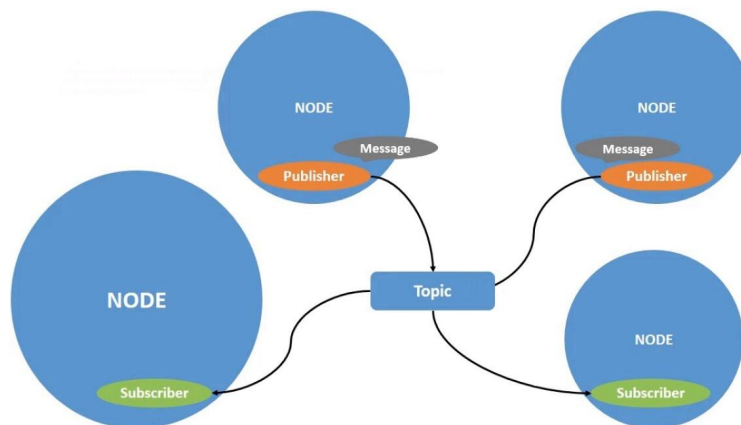


图 3-2-6 节点与话题之间的关系

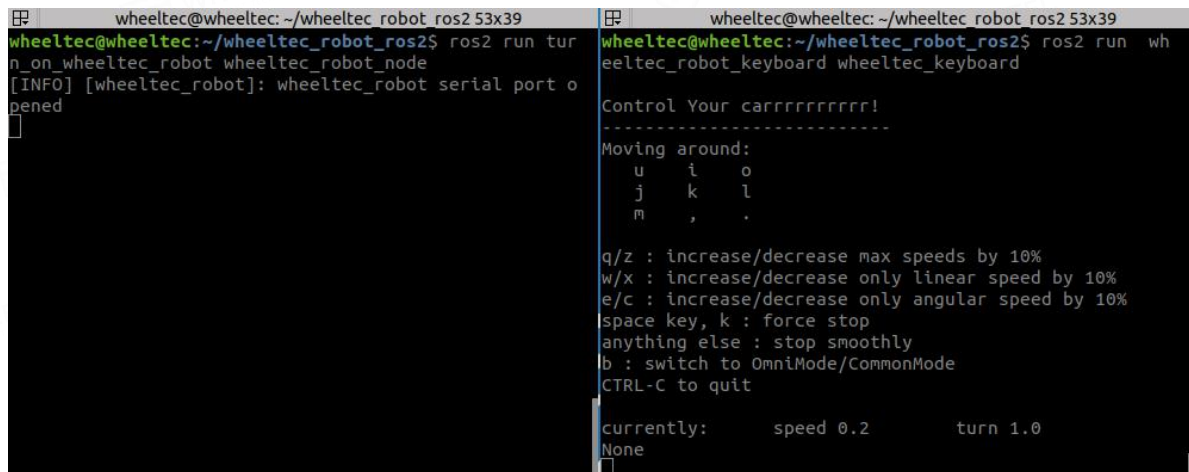
以下以打开 wheeltec_robot 节点与键盘控制节点为示例，简单说明话题相关

指令的使用方式。注：执行指令前终端需要先调用 ROS2 的系统环境。

```
initros2
cd ~/wheeltec_robot_ros2
sh install/setup.bash
```

1) ROS2 运行执行例程指令：

```
#启动初始化底盘控制
ros2 run turn_on_wheeltec_robot wheeltec_robot_node
#程序包名称为 turn_on_wheeltec_robot，可执行文件名称为 wheeltec_robot_node
#打开键盘控制节点
ros2 run wheeltec_robot_keyboard wheeltec_keyboard
```



The image shows two terminal windows. The left window shows the execution of ROS2 commands to start the robot node and the keyboard node. The right window shows the output of the keyboard node, displaying a control interface with movement instructions and current speed/turn values.

图 3-2-6 运行示例

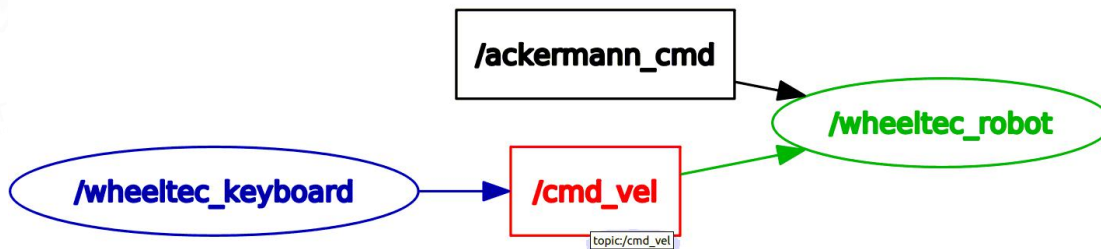


图 3-2-7 节点之间的可视化信息高亮显示

如果将鼠标悬停在中间的话题上，我们可以看到颜色高亮显示，如上图所示。该图描述了 wheeltec_keyboard 节点和 wheeltec_robot 节点如何通过 /cmd_vel 话题相互通信。wheeltec_keyboard 节点正在将数据发布到 /cmd_vel 话题，wheeltec_robot 节点订阅 /cmd_vel 话题以接收数据。

当检查具有许多节点和主题以许多不同方式连接的更复杂的系统时，rqt_graph 的突出显示功能非常有用。

- 2) 使用 `ros2 topic list` 查看系统中当前活动的所有话题的列表，在指令后添加 `-t` 参数会返回话题的消息数据类型，如图 3-2-8 所示

```
wheeltec@wheeltec:~$ ros2 topic list
/PowerVoltage
/ackermann_cmd
/cmd_vel
/mobile_base/sensors/imu_data
/odom_combined
/parameter_events
/robotpose
/robotvel
/rosout
/tf
wheeltec@wheeltec:~$ ros2 topic list -t
/PowerVoltage [std_msgs/msg/Float32]
/ackermann_cmd [ackermann_msgs/msg/AckermannDriveStamped]
/cmd_vel [geometry_msgs/msg/Twist]
/mobile_base/sensors/imu_data [sensor_msgs/msg/Imu]
/odom_combined [nav_msgs/msg/Odometry]
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/robotpose [wheeltec_robot_msgs/msg/Data]
/robotvel [wheeltec_robot_msgs/msg/Data]
/rosout [rcl_interfaces/msg/Log]
/tf [tf2_msgs/msg/TFMessage]
wheeltec@wheeltec:~$
```

图 3-2-8 系统中所有话题列表及其消息数据类型

- 3) 使用 `ros2 topic echo` 查看有关某个主题的发布数据，指令格式为：

```
ros2 topic echo <topic_name>
```

```
wheeltec@wheeltec:~$ ros2 topic echo /cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
```

图 3-2-9 使用 `ros2 topic echo` 查看/cmd_vel 话题的发布数据

- 4) 话题并不仅限于点对点交流；它可以是一对多，多对一或多对多。使用 `ros2 topic info` 查看话题的详细信息，指令格式为：

```
ros2 topic info <topic_name>
```

```
wheeltec@wheeltec:~$ ros2 topic info /cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscriber count: 1
wheeltec@wheeltec:~$
```

图 3-2-10 使用 `ros2 topic info` 查看/cmd_vel 话题信息

- 5) 使用 `ros2 topic pub` 从命令行将数据发布到话题中。节点使用消息通过主

题发送数据。发布者和订阅者必须发送和接收相同类型的消息才能进行通信。在上一步可以查看到，/cmd_vel 这个话题的消息类型是 geometry_msgs/msg/Twist, 有了消息结构后，我们可以使用 ros2 topic pub 直接从命令行将数据发布到主题上。指令格式为：

```
ros2 topic pub <topic_name> <msg_type> '<args>'
```

'<args>'这个参数将要传递给话题的实际数据。以发布速度话题为例：

```
wheeltec@wheeltec:~$ ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 1.0, y: 0.2, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #2: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #3: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #4: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #5: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #6: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #7: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
publishing #8: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=1.0, y=0.2, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
^Cwheeltec@wheeltec:~$
```

图 3-2-11 使用 ros2 topic pub 发布数据到/cmd_vel 话题

```
---
linear:
  x: 1.0
  y: 0.2
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

图 3-2-11 查看/cmd_vel 话题数据

6) 使用 ros2 topic hz 查看话题发布速率，指令格式为：

```
ros2 topic hz <topic_name>
```

```
wheeltec@wheeltec:~$ ros2 topic hz /cmd_vel
average rate: 10.041
  min: 0.089s max: 0.101s std dev: 0.00324s window: 11
average rate: 9.990
  min: 0.089s max: 0.101s std dev: 0.00241s window: 21
average rate: 9.966
  min: 0.089s max: 0.103s std dev: 0.00206s window: 31
```

图 3-2-11 查看/cmd_vel 话题发布频率

③ 其它常用指令

1) ROS2 service 的相关指令：

表 3-3-1 ROS2 service 的相关指令信息

指令内容	指令功能说明
ros2 service list	查看服务列表
ros2 srv show	显示所有服务类型信息
ros2 service find	根据服务类型寻找当前服务
ros2 service call	以命令行的形式调用服务，可以指定具体参数，指令格式为： <pre>ros2 service call <service_name> <service_type> <arguments></pre>

2) ROS2 param 的相关指令

表 3-3-2 ROS2 param 的相关指令信息

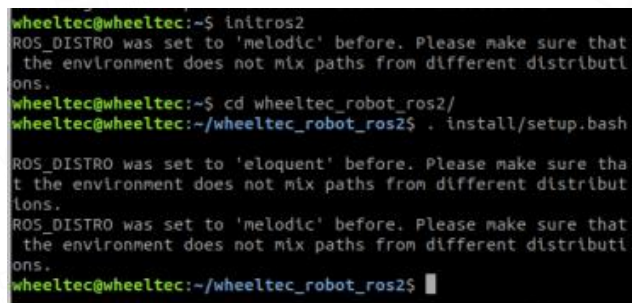
指令内容	指令功能说明
ros2 param list	列出参数服务器中的参数
ros2 param get <pre>ros2 param get <node_name> <parameter_name></pre>	获取参数值
ros2 param set <pre>ros2 param set <node_name> <parameter_name> <value></pre>	设置参数值
ros2 param delete	删除参数
ros2 param dump <pre>ros2 param dump <node_name></pre>	将参数服务器中的参数写入到文件。 加载配置文件 <pre>ros2 run <package_name> <executable_name> --ros-args --params-file <file_name></pre>

3.3 在 wheeltec_robot 中使用 ROS2

本章是基于 wheeltec_robot 机器人做关于 ROS2 程序使用层面的讲解，方便用户迅速跑通 ROS2。如果用户要为自己的项目编写特定的功能包，推荐完整学习 ROS2 中的[用户手册](#)，其中包含更加全面的指导信息。

在运行程序之前需要先调出 ROS2 的环境，并声明工作空间的具体位置。终端依次执行以下指令：

```
initros2 #调出 ROS2 环境
cd ~/wheeltec_robot_ros2
. install/setup.bash
```



```
wheeltec@wheeltec:~$ initros2
ROS_DISTRO was set to 'melodic' before. Please make sure that
the environment does not mix paths from different distributions.
wheeltec@wheeltec:~$ cd wheeltec_robot_ros2/
wheeltec@wheeltec:~/wheeltec_robot_ros2$ . install/setup.bash
ROS_DISTRO was set to 'eloquent' before. Please make sure that
the environment does not mix paths from different distributions.
ROS_DISTRO was set to 'melodic' before. Please make sure that
the environment does not mix paths from different distributions.
wheeltec@wheeltec:~/wheeltec_robot_ros2$
```

图 3-3-1 初始化工作环境

① 关于车型选择

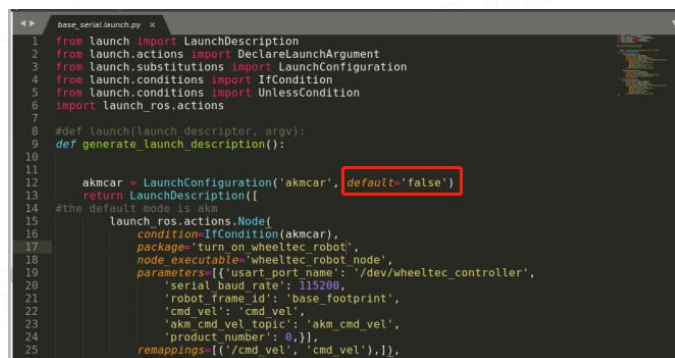
由于 Ackerman 机器人的转向结构和其它车型的转向结构不一致，所以若用户使用的是阿克曼小车，则需要在以下目录做出修改。

```
/wheeltec_robot_ros2/src/turn_on_wheeltec_robot/launch/base_serial.launch.py
```

将目录中的 base_serial.launch.py 文件中相应的位置改为”true”，非阿克曼小车的 base_serial.launch.py 文件中 akmcarr 的值为”false”。如图 3-3-2 所示。

注意：在 ROS2 中修改完 python 文件也是需要进行编译才能生效的，所以这里更改完之后需要执行以下指令编译一遍。

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



```
1 from launch import LaunchDescription
2 from launch.actions import DeclareLaunchArgument
3 from launch.substitutions import LaunchConfiguration
4 from launch.conditions import IfCondition
5 from launch.conditions import UnlessCondition
6 import launch_ros.actions
7
8 def launch(launch_description, argv):
9     def generate_launch_description():
10
11         akmcarr = LaunchConfiguration('akmcarr', default='false')
12         return LaunchDescription([
13             #the default mode is ack
14             launch_ros.actions.Node(
15                 condition=IfCondition(akmcarr),
16                 package='turn_on_wheeltec_robot',
17                 node_executable='wheeltec_robot_node',
18                 parameters=[{'uart_port_name': '/dev/wheeltec_controller',
19                             'serial_baud_rate': 115200,
20                             'robot_frame_id': 'base_footprint',
21                             'cmd_vel': 'cmd_vel',
22                             'ack_cmd_vel_topic': 'ack_cmd_vel',
23                             'product_number': 0}],
24                 remappings=[('/cmd_vel', 'cmd_vel')]),
25
26
```

图 3-3-2 base_serial.launch.py 文件

② turn_on_wheeltec_robot

turn_on_wheeltec_robot 这个功能包的主要作用是打开初始化底盘控制节点，运行指令为：

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

这里需要注意 ros2 和 launch 之间的空格。ROS2 系统中，运行 ROS2 的相关指令时，在 ROS2 后面需加上一个空格。

在 turn_on_wheeltec_robot.launch.py 文件中，调用了 wheeltec_robot 机器人上的传感器坐标与机器人的 base_footprint 的 tf 坐标转换、关于机器人描述的 urdf 文件以及融合滤波的节点。

③ simple_follower_ros2

在 simple_follower 这个功能包中有两个启动文件，实现的功能是视觉巡线与雷达跟随。

运行视觉巡线指令为：

```
ros2 launch simple_follower_ros2 line.launch.py
```

这里需要注意 ros2 和 launch 之间的空格。

在 line.launch.py 文件中打开了 usb 摄像头和初始化底层控制节点，所以用户不必另外再运行摄像头和 turn_on_wheeltec_robot.launch.py 底盘启动文件。

运行雷达跟随指令为：

```
ros2 launch simple_follower_ros2 laser_follow.launch.py
```

这里需要注意 ros2 和 launch 之间的空格。

在 line.launch.py 文件中打开了雷达和初始化底层控制节点，所以用户不必另外再运行 turn_on_wheeltec_robot.launch.py 底盘启动文件。

④ wheeltec_robot_slam

wheeltec_robot_slam 功能包主要是实现 2D 建图功能。文件夹里面包含了三个 slam 功能包，分别是 slam_gmapping、slam_toolbox 和 cartographer 功能包的调用。

slam_gmapping 软件包包含用于 OpenSlam 的 Gmapping 的 ROS 包装器，提供了基于激光的 SLAM（同时定位和映射），作为 slam_gmapping 的 ROS 节点，它使用的是 Gmapping 的绘制地图算法。节点 slam_gmapping 订阅了 ros2 topic 上的 sensor_msgs / LaserScan 雷达话题数据/scan。

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。

cartographer 是在 ROS1 中也常用到的一种建图算法，这里就不再赘述。

建图步骤为：用户可以挑选以下三种建图方式的任意一种进行 2D 建图，使用键盘控制节点或蓝牙 APP 或航模遥控控制小车运动从而完成建图。

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
```

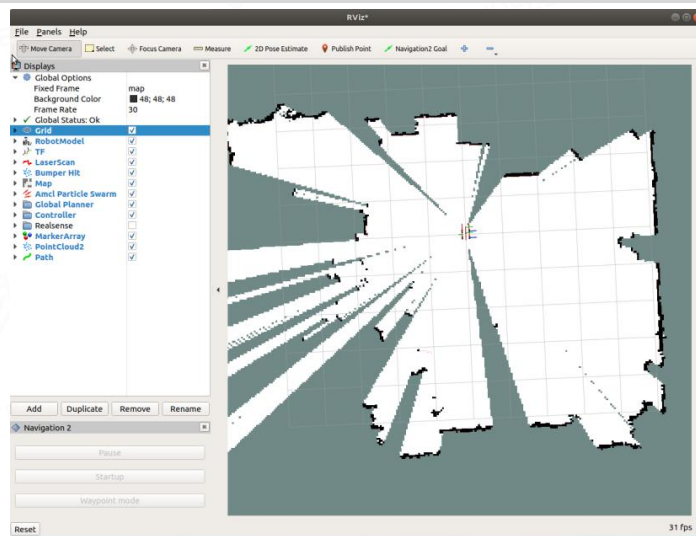


图 3-3-3 gmapping 算法建图页面

使用 slam_toolbox 进行 2D 建图，指令如下：

```
ros2 launch wheeltec_slam_toolbox online_async_launch.py
```

建图页面如图 3-2-4 所示。

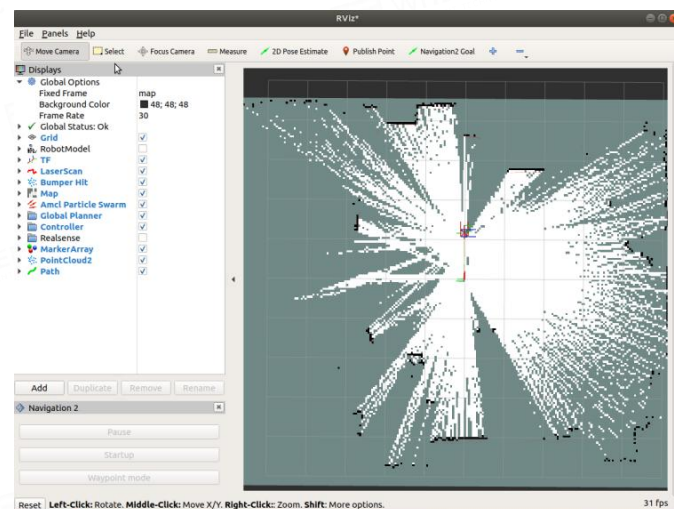


图 3-3-4 使用 slam_toolbox 建图页面

使用 cartographer 算法进行 2D 建图，指令如下。建图页面如图 3-2-5 所示。

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

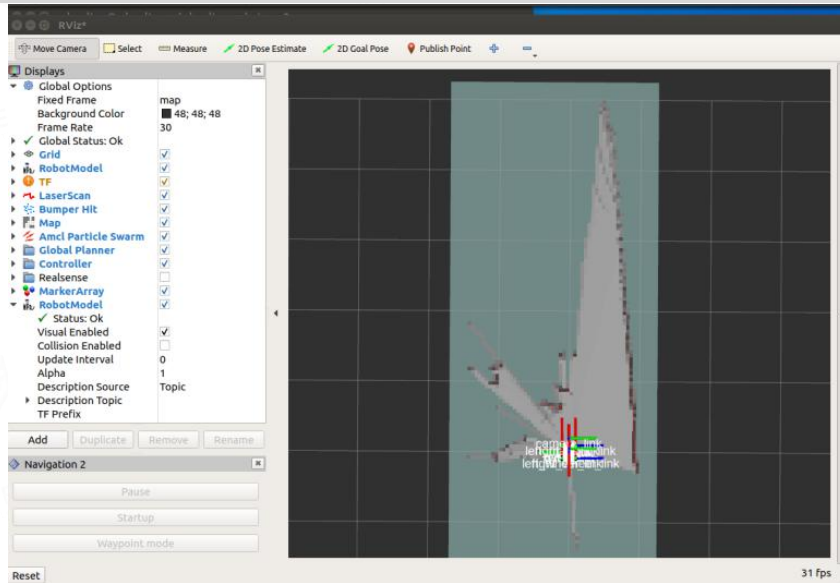


图 3-3-5 使用 slam_toolbox 建图页面

1) 打开键盘控制节点控制机器人运动：

```
ros2 run wheeltec_robot_keyboard wheeltec_keyboard
```

2) 打开 rviz2 可视化工具：

```
rviz2
```

3) 建图完成，使用以下指令保存地图：

```
save_map
```

⑤ wheeltec_robot_nav

1) 2D 单点导航

wheeltec_robot_nav 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。

在 wheeltec_navigation.launch.py 文件中，调用了初始化底层控制节点、导航的地图(地图文件存放在 wheeltec_robot_nav/map 目录下)、以及导航行为树的相关参数(参数文件存放在 wheeltec_robot_nav/param 目录下)以及 localization 功能包。

在 wheeltec_navigation.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。确保 use_sim_time 设置为 False，因为我们要使用系统时间而不是 Gazebo 中的时间模拟时间。

执行 2D 导航的指令为：

```
ros2 launch wheeltec_robot_nav wheeltec_navigation.launch.py
```


执行完以上指令后，打开一个终端，输入指令：

```
rviz2
```

打开可视化页面，点击 rviz2 上方的”Navigation2 Goal”设置目标点进行导航。

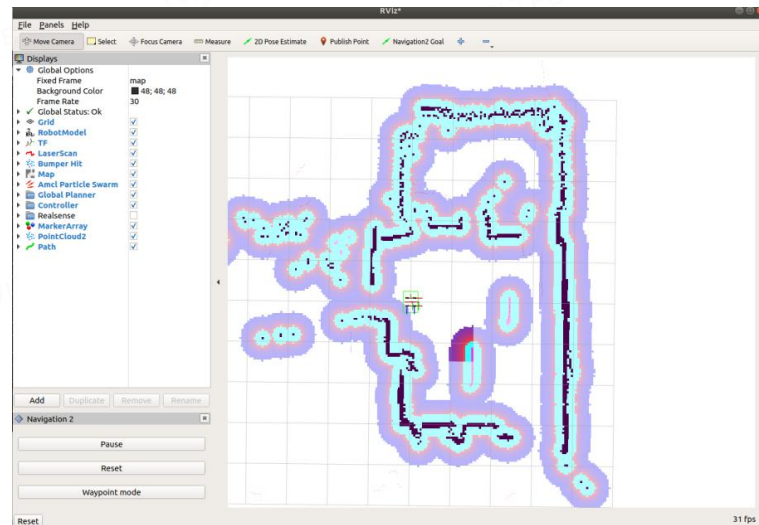


图 3-3-6 rviz2 可视化导航页面

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，点击右下角的”waypoint mode”切换到路标跟随模式，如图 3-2-7 所示。

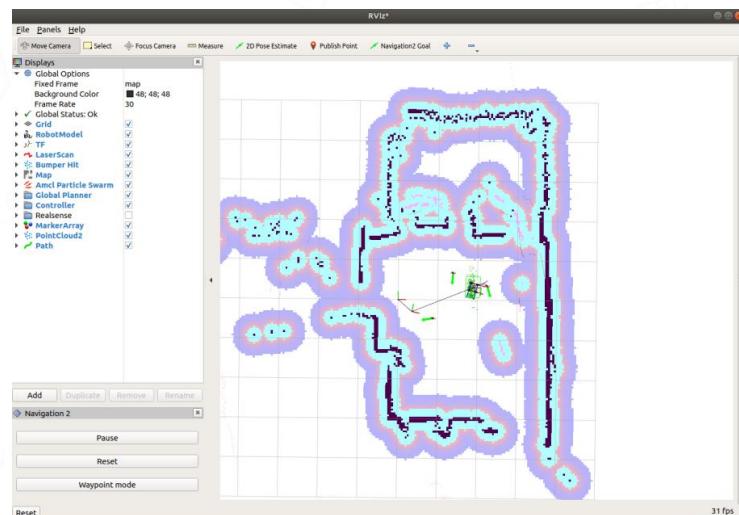


图 3-3-7 waypoint mode 设置页面

到这一步，再点击 rviz2 上方的”Navigation2 Goal”开始设定路标位置，标完位置后，点击右下角的”stat”开始实现路标跟随。

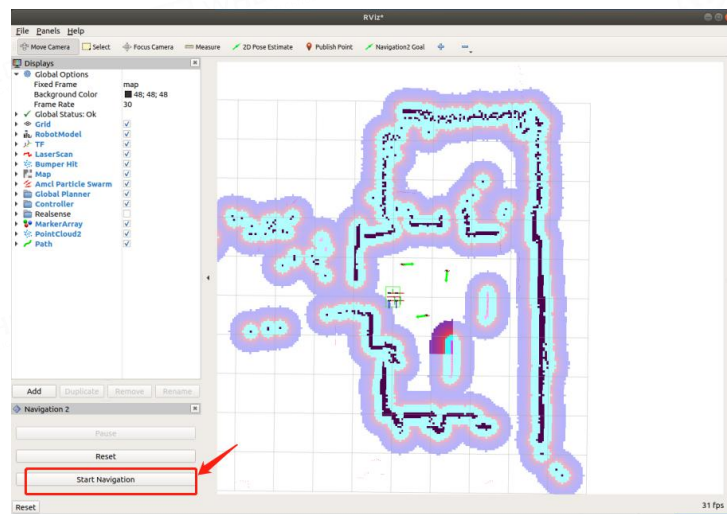


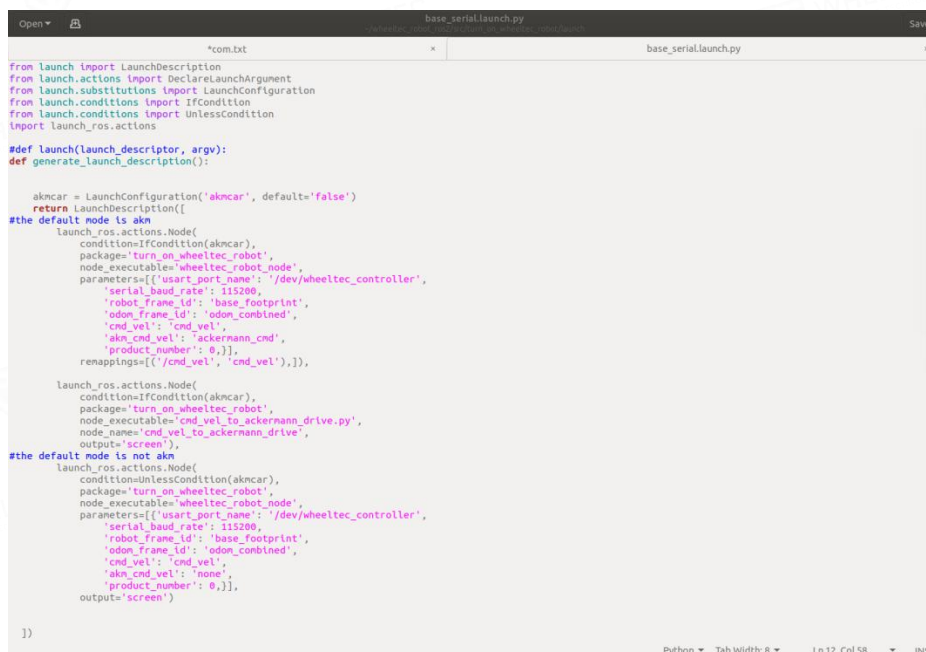
图 3-3-7 waypoint mode 开始导航页面

3.4 launch 启动文件说明

当用户创建越来越多的节点同时运行的更复杂的系统时，打开终端并重新输入配置详细信息将变得很繁琐。启动文件可以同时启动和配置许多包含 ROS 2 节点的可执行文件。ROS1 中启动文件使用在 XML 语法编写，功能性比较有限。在 ROS 2 中，`roslaunch` 是以 Python 编写，因此可以使用像是条件判断式等比较复杂的逻辑。

启动的主要对象是类别：`LaunchDescriptionEntity`，启动的其他实体继承。用户的目的是如何启动系统，以及启动过程中如何对系统中的异步事件作出反应。其中一个特殊实体名为：`LaunchDescription.LaunchDescription` 以启动启动过程。这个类将用户的意图封装为离散的列表。作为启动描述实体本身，这些操作通常是为了响应启动系统中的事件。

本节以 `turn_on_wheeltec_robot` 功能包中的 `base_serial.launch.py` 为示例，分析如何编写 ROS2 中的启动文件。`base_serial.launch.py` 文件如图 3-4-1 所示。



```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch.conditions import IfCondition
from launch.conditions import UnlessCondition
import launch_ros.actions

def launch(launch_descriptor, argv):
    def generate_launch_description():

        akmcar = LaunchConfiguration('akmcar', default='false')
        return LaunchDescription([
            #the default node is akm
            launch_ros.actions.Node(
                condition=IfCondition(akmcar),
                package='turn_on_wheeltec_robot',
                node_executable='wheeltec_robot_node',
                parameters=[('uart_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': 'ackermann_cmd',
                    'product_number': 0)],
                remappings=[('/cmd_vel', 'cmd_vel')]),

            launch_ros.actions.Node(
                condition=IfCondition(akmcar),
                package='turn_on_wheeltec_robot',
                node_executable='cmd_vel_to_ackermann_drive.py',
                node_name='cmd_vel_to_ackermann_drive',
                output='screen'),

            #the default node is not akm
            launch_ros.actions.Node(
                condition=UnlessCondition(akmcar),
                package='turn_on_wheeltec_robot',
                node_executable='wheeltec_robot_node',
                parameters=[('uart_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 3-4-1 `base_serial.launch.py` 文件

首先是使用 `import` 语句引入了一些 Python `launch` 模块。

接下来，定义一个描述变量 `akmcar` 用于车型的切换；

启动描述本身开始：

```
def generate_launch_description():
```

```
return LaunchDescription([  
    ])
```

内部 `LaunchDescription` 是由两个个节点组成的系统，这个两个节点均来自 `turn_on_wheeltec_robot` 软件包。该文件的目标是启动机器人的底层初始化节点 `wheeltec_robot_node`。注：这里加载的参数可以是具体数值也可以是参数文件。

```
Node(  
    package='turn_on_wheeltec_robot', #功能包名  
    node_executable='wheeltec_robot_node', #可执行节点名  
    parameters=[{'usart_port_name': '/dev/wheeltec_controller', #加载参数数据  
                  'serial_baud_rate': 115200,  
                  'robot_frame_id': 'base_footprint',  
                  'odom_frame_id': 'odom_combined',  
                  'cmd_vel': 'cmd_vel',  
                  'akm_cmd_vel': 'ackermann_cmd',  
                  'product_number': 0,}],  
    remappings=[('/cmd_vel', 'cmd_vel'),], #重映射话题信息
```

条件的概念是一个谓词，适用于某些种类的实体，例如（但不仅限于）`launch.Action`。这与 ROS 1 中的 `if` 和 `unless` 属性相同。

拟议的 `launch.Condition` 将取一个条件变量，该变量将是一个替换列表。访问关联实体时将替换列表转换为字符串，计算结果条件变量字符串为 `true / false`。

条件必须成为任何可能受其影响的实体的接口的一部分。以下这段代码包含了一个条件选择。在文件中可以看到，默认 `akmcar` 这个变量是 `false`，则整个 `base_serial.launch.py` 文件执行以下这一部分代码

```
#the default mode is not akm  
launch_ros.actions.Node(  
    condition=UnlessCondition(akmcar),  
    package='turn_on_wheeltec_robot',  
    node_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 3-4-2 `base_serial.launch.py` 文件

若默认 `akmcar` 这个变量是 `true`，则 `condition=IfCondition(akmcar)`生效，`base_serial.launch.py` 文件执行以下这一部分代码。

```
#the default mode is akn
launch_ros.actions.Node(
    condition=IfCondition(akmcar),
    package='turn_on_wheeltec_robot',
    node_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': 'ackermann_cmd',
                  'product_number': 0,}],
    remappings=[('/cmd_vel', 'cmd_vel')]),

launch_ros.actions.Node(
    condition=IfCondition(akmcar),
    package='turn_on_wheeltec_robot',
    node_executable='cmd_vel_to_ackermann_drive.py',
    node_name='cmd_vel_to_ackermann_drive',
    output='screen'),
```

图 3-4-3 base_serial.launch.py 文件

4. 由 ROS1 过度到 ROS2

4.1 ROS1 与 ROS2 之间的差异

前文已经提到过 ROS1 与 ROS2 之间存在的差异，现在在这一节中来总结一下具体有哪些差异。

① 平台及相依性

ROS 1 仅在 Ubuntu 上做持续整合测试,其他 Linux 平台及 OS X 的支援由社群资源提供。

ROS 2 目前支持 Ubuntu Xenial, OS X El Capitan 以及 Windows 10 等平台,并在上述平台上做持续整合测试。

② 程序语言

ROS 1 以 C++03 为其核心,在其 API 并没有利用 C++11 的功能。

ROS 2 则广泛运用了 C++11 标准,并部分运用了 C++14 的版本功能; ROS 1 主要支持 Python 2, 而 ROS 2 则支援 Python 3.5 以上版本。

③ 通讯系统

ROS 1 的通讯系统基于 TCPROS/UDPROS, 强依赖于 master 节点的处理。

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。

④ 编译系统

所有的 ROS package 皆可视为一个 CMake 项目。ROS 2 也支援其他编译系统。目前除 CMake 外, 编译工具也支援单纯的 Python package, 关于构建系统

ament_cmake 的更多信息请看 [ament](#)

⑤ Python 功能包

ROS 1 中, `setup.py` 文件是以 CMake 的自定义逻辑处理, 因此包含 Python 程式码的 ROS 功能包只能使用一部分 `setup.py` 文件中的功能。

在 ROS 2 中, 由于 Python package 可以通过 `python3` 的 `setup.py install` 调用, 因此可完整地使用 `setup.py` 文件。

⑥ 环境配置

在 ROS 1 中, 在使用编译完成的 ROS 包前, 必须通过 `source` 指令使用编译工具产生的脚本(scripts)来配置运行环境。这种方法只在使用 ROS 特定构建工具构建 ROS 包时才有效。

在 ROS 2 中, 环境设置分为特定于包的脚本和特定于工作区的脚本。每个包都提供了必要的脚本, 使其在构建后可以使用。构建工具只调用特定于工作区的脚本, 然后调用特定于包的脚本。

⑦ 没有非独立编译

在 ROS 1 中, 多个功能包可在单一个 CMake 文件下编译, 这可以让编译的速度更快, 但每个包都需要确保正确定义跨包目标依赖项。此外, 所有包都共享同一个名称空间, 这会出现目标名称发生冲突等问题。

在 ROS 2 中, 只支持独立的编译, 也就是说, 每个包都单独编译。而安装的空间可以是独立或是合并的。

⑧ 没有 devel space

在 ROS 1, 功能包可以在没有安装的情形下编译, 从开发空间和源空间相结合, 系统已经是可用的。但是每个包都必须积极地支持开发空间, 例如在环境挂钩和 CMake 代码中。

而在 ROS 2, 功能包在编译后必须先安装, 才能使用。

⑨ 支持使用 `catkin_simple`

在 ROS 1 中, `catkin_simple` 的目的在于简化 ROS package 中的 CMake 程序编写。但在很多时候, 由于要符合一些像是支持 `devel space` 功能的设计需求限制下, `catkin_simple` 并没有达到原有的目的。

在 ROS 2, 则针对这样的使用案例, 重新架构了 CMake API。

⑩ 对于无描述资料功能包的基本支持

在 ROS 1 中, 构建系统只考虑包含清单文件的包。在 ROS 2 中, 可以在没

有清单文件的文件夹中检测到支持构建系统的包。如果包遵循常规做法,甚至可能侦测出一些遗漏的描述资料(如依赖项)。

⑪ C++命名空间的划分

在 ROS 1, .msg 及 .srv 档案可以有相同的文件名,但生成的代码会发生冲突。相同的情形也会发生在 services 的 request 及 response 程序中。

在 ROS 2, 生成的代码会划分彼此的命名空间来避免冲突。但目前由于 DDS 有效字符的限制, ROS2 并没有支持话题(Topic)的命名空间。

⑫ python 中的同名

为消息和服务生成的 Python 代码目前在 ROS 1 和 ROS 2 中使用相同的模块和类名,因此不能在单个应用程序中同时导入它们。

⑬ 客户端库的变化

- 1) ROS 1 中关于 ROS graph 的资讯都要通过 master 获取。而 ROS 2 可通过 publish 的方式针对变动发出通知。
- 2) 在 ROS 1 中每个节点都有其主功能。。而 ROS 2 则是建议从具有生命周期的组件中继承,这样的周期可便于类似 roslaunch 类型的工具来启动多个节点组成的系统。详情请看 [node_lifecycle](#)
- 3) 在 ROS 1 中,全局参数和特定于节点的动态重新配置参数是两个独立的概念。在 ROS 2 中,正在使用统一的方法。它类似于动态重新配置,名为“全局参数服务器”的节点将无条件地接受设置值的请求。在 ROS 1 中,所有这些信息都需要对 ROS 2 中的更改进行轮询,更改将发布以通知其他实体。详情请看 [ros_parameters](#)
- 4) ROS 1 并不支持实时控制的程序代码,而是依赖类似 Orocos 的外部架构来达到用户要求。而在 ROS 2 中,如果使用适合的 RTOS 并有周详的软体搭配,是可以实现实时控制节点的。关于实时系统的更多信息请看: [realtime](#)
- 5) ROS 1 中 nodes 及 nodelets 的 API 是不同的,用户必须在编写程序时决定 nodes 与执行程序的对应关系。在 ROS 2 中,取消了 nodelets 的概念。
- 6) ROS 1 中,在单一程序中不能有多个 nodes. 这导因于 API 本身及内部建置时的决定。而在 ROS 2 则允许单一程序里创建多个 nodes。

- 7) 在 ROS 1 中, roslaunch 档在 XML 中定义, 功能性比较有限。在 ROS 2 中, roslaunch 是以 Python 编写, 因此可以使用像是条件判断式等比较复杂的逻辑。
- 8) ROS 1 中不同的资源(packages, messages, plugins...等)是基于 ROS_PACKAGE_PATH 来查找文件系统。而当 ROS_PACKAGE_PATH 太庞大时, 这样的做法会造成系统的效能低落。在 ROS 2 中, 资源会在编译时生成索引, 而在执行时有效地查询。更多资讯可参照 [resource index](#)

4.2 ROS1 与 ROS2 的通信

到目前为止, ROS2 仍处于开发之中, 还没有一个稳定的长期支持的发行版。因此, 如果直接将 ROS1 中已有的节点程序移植到 ROS2 下将是一项艰巨的任务。为了使用户能够同时使用 ROS1 和 ROS2 的功能包进行开发, 可以通过一个 rosl_bridge 的工具实现 ROS1 与 ROS2 的通信, 从而实现两个版本的连接与共用。

① rosl_bridge 介绍

rosl_bridge 是 ROS2 的一个功能包, 用于自动或手动建立信息, 话题或者服务的映射, 并在 ROS1 和 ROS2 的节点之间进行通信。

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 rosl_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。

② 通信过程

使用步骤示例:

- 1) 在 ROS1 中运行 roscore

```
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 rosl_bridge

```
initros2
```

```
ros2 run rosl_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。从 ROS2 到 ROS1 的话题通信步骤为：

1) 在 ROS1 中运行 roscore

```
roscore
```

2) 在 ROS1 的终端打开键盘控制节点

```
roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

3) 在 ROS2 的终端打开 ros1_bridge

```
initros2
```

```
ros2 run ros1_bridge dynamic_bridge
```

4) 在 ROS2 的终端打开底盘控制节点

```
initros2
```

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

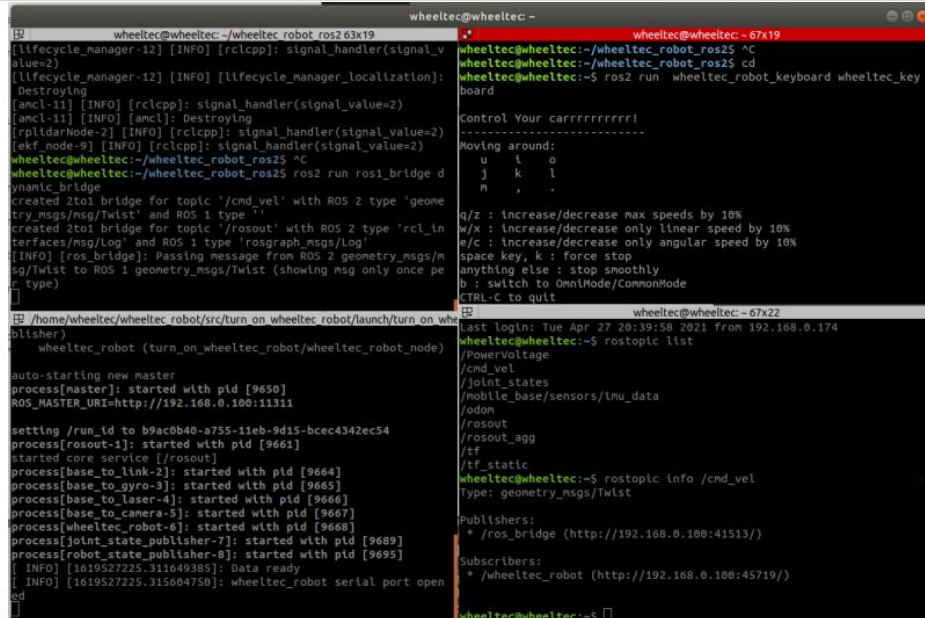


图 3-3-1 ROS1 与 ROS2 之间的服务通信