



Mobile Robot Vision Expert

Linux 示例程序 使用说明书

Linux

杭州蓝芯科技 & MRDVS 有限公司 版权所有

2024.04.02

目录

一、概述	1
二、系统与环境	2
2.1 系统与环境介绍	2
2.2 查看系统版本与应用版本	2
2.2.1 查看系统版本	2
2.2.2 查看 Cmake 版本	2
2.2.3 查看 Gcc 版本	3
2.2.4 查看 Opencv 版本	3
2.2.5 查看 Pcl 版本	3
2.2.6 查看 ROS 版本	3
2.3 搭建基础开发环境	4
2.3.1 安装 Cmake	4
2.3.2 安装 Gcc	4
2.4 搭建可选开发环境	4
2.4.1 安装 Opencv 【可显示相机图像】	4
2.4.2 安装 Pcl 【可显示点云】	4
三、示例代码的介绍与运行准备	5
3.1 执行安装脚本	5
3.2 示例代码介绍	5
3.2.1 C 与 C++ 示例代码	6
3.2.2 Java 示例代码	6
3.2.3 ROS 示例代码	7
3.2.4 ROS2 示例代码	7
3.2.5 ros-v1pro 示例代码	7
四、示例代码的编译与运行	8
4.1 C 与 C++ 示例代码的编译与运行	8
4.2 java 示例运行流程	9
4.3 ROS 示例代码的编译与运行	10
4.4 ROS2 示例代码的编译与运行	11
4.5 ros-v1pro 示例代码的编译与运行	12
五、示例代码解释与说明	13
5.1 C 与 C++	13
5.1.1 CMakeLists	13
5.1.2 application_obstacle 和 application_pallet	14
5.1.3 arm_local_camera	14
5.1.4 frame_callback	15
5.1.5 multi_cameras	15
5.1.6 single_camera	16
5.1.7 single_camera2	17
5.2 java	17
5.3 ROS1 & ROS2	20
5.3.1 ROS1 launch 文件说明	20

5.3.2 ROS2 launch 文件说明	24
5.3.3 特有主题说明	28
5.3.4 特有服务说明	32
5.3.5 主题订阅方式说明	34
5.3.6 服务通讯方式说明	43
六、ROS 推荐硬件配置与资源占用	55
6.1 推荐硬件配置	55
Ubuntu16.04	55
Ubuntu18.04	55
Ubuntu20.04	55
Ubuntu22.04	55
6.2 CPU 详情	56
Ubuntu16.04	56
Ubuntu18.04	57
Ubuntu20.04	58
Ubuntu22.04	59
6.3 内存详情	60
Ubuntu16.04	60
Ubuntu18.04	61
Ubuntu20.04	62
Ubuntu22.04	63
附录:	64
1. Cmake 指定版本安装方法	64
2. Opencv 指定版本安装方法	64
3. Pcl 指定版本安装方法	65
4. 安装 ROS	65
Ubuntu16.04	65
Ubuntu18.04	66
Ubuntu20.04	67
Ubuntu22.04	68
5. Ros 如何创建一个新的测试包	68
6. 安装 ROS2	70
Ubuntu16.04	70
Ubuntu18.04:	71
Ubuntu20.04	72
Ubuntu22.04	74
7. Ros2 如何创建一个新的测试包	75

一、概述

【蓝芯自研相机 Linux 开发包】提供了多种示例程序，包括：C、JAVA(http 服务器)，ros 和 ros2。示例代码提供 SDK 的使用方法，以及一些基本的功能、方法，但具体功能的实现仍需要手动修改。本文主要介绍各个示例程序的用法、需要搭建的环境、以及一些注意事项。以下会用少量的篇幅介绍 JAVA、C、C++相关的代码功能介绍，用中量篇幅介绍环境相关的问题，用大量篇幅介绍 ROS 与 ROS2 相关的用法、环境、注意事项等。

推荐优先顺序阅读第一至第三部分，在顺序阅读第一至第三部分之后，对于不同的需求，可以直接跳转到相关部分阅读，各个章节互相独立，不影响查阅。

最后，本文提供一些相关应用与环境的安装方法与注意事项，如有需要可以查阅附录。

二、系统与环境

2.1 系统与环境介绍

推荐使用以下几种 linux 系统，默认的程序版本见下表：

系统版本	Cmake/Gcc	Opencv	Pcl	Ros	Ros2
Ubuntu16.04	3.5 / Gcc-5	3.2.0	1.7.2	Kinetic	Ardent、Bouncy
Ubuntu18.04	3.18 / Gcc-7	4.5.1	1.9.1	Melodic	Dashing、Eloquent Crystal
Ubuntu20.04	3.22 / 9.3.0	4.5.1	1.12.1	Noetic	Foxy、Galactic
Ubuntu22.04	3.22 / 11.3.0	4.6.0	1.12.1	Ninjemys	Humble、Iron

2.2 查看系统版本与应用版本

2.2.1 查看系统版本

命令：uname -a

结果：Linux ubuntu 5.15.0-97-generic #107~20.04.1-Ubuntu

操作系统发行版本号：5.15.0

- ❖ 主版本号：5
- ❖ 次版本号：15【奇数为开发版本，偶数为稳定版本】
- ❖ 修订版本号：0【修改的次数】
- ❖ 此版本的第 N 次修改：97
- ❖ 内核版本：#107~20.0.4.1，通常所说的系统版本就是内核版本

2.2.2 查看 Cmake 版本

命令：cmake -version

结果：cmake version 3.16.3

代表 Cmake 的版本为 3.16.3

2.2.3 查看 Gcc 版本

命令：gcc -v

结果：

```
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:hsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 9.4.0-1ubuntu1-20.04.2' --with-bugurl=file:///usr/share/doc/gcc-9/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,gn2 --prefix=/usr --with-gcc-major-version-only --program-suffix=-9 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib=auto --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none=/build/gcc-9-9QD0t0/gcc-9-9.4.0/debian/tmp-nvptx/usr,hsa --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1-20.04.2)
```

版本号在左下角的蓝色框内，此处版本为：9.4.0

2.2.4 查看 Opencv 版本

命令：pkg-config --modversion opencv

结果：

```
fr1511b@ubuntu:~/Desktop/Lanxin-MRDVS/Sample/ROS2$ pkg-config --modversion opencv
3.2.0
```

此处版本为 3.2.0

2.2.5 查看 Pcl 版本

命令：dpkg -l libpcl-dev

结果：

```
fr1511b@ubuntu:~/Desktop/Lanxin-MRDVS/Sample/ROS2$ dpkg -l libpcl-dev
期望状态=未知(u)/安装(i)/删除(r)/清除(p)/保持(h)
| 状态=未安装(n)/已安装(i)/仅存配置(c)/仅解压缩(u)/配置失败(F)/不完全安装(H)/触发器等待(W)/触发器未决(T)
|/ 错误?=(无)/须重装(R) (状态, 错误: 大写=故障)
||/ 名称 版本 体系结构 描述
+++-----+-----+-----+-----+
ii libpcl-dev 1.8.1+dfsg1-2ubun amd64 Point Cloud Library - development files
```

此处版本为 1.8.1

2.2.6 查看 ROS 版本

命令：rosversion -d

结果：

```
fr1511b@ubuntu:~/Desktop/Lanxin-MRDVS/Sample/ROS2$ rosversion -d
eloquent
```

此处版本为 eloquent (Ubuntu18.04 对应的 ros2 版本)

```
fr1511b@ubuntu:~/Desktop/Lanxin-MRDVS/Sample/ros/lx_camera_node_ws$ rosversion -d
noetic
```

此处版本为 noetic (Ubuntu20 对应的 ros1 版本)

2.3 搭建基础开发环境

在安装之前需要先更新安装列表：

执行命令：sudo apt update

结果如下：

```
fr1511b@ubuntu:~/Desktop/Lanxin-MRDVS/Sample/ros/lx_camera_node_ws$ sudo apt update
[sudo] password for fr1511b:
Hit:2 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal InRelease
Hit:1 https://packages.microsoft.com/repos/code stable InRelease
Hit:3 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-updates InRelease
Hit:4 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal-backports InRelease
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:6 http://packages.ros.org/ros/ubuntu focal InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
124 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

2.3.1 安装 Cmake

命令：sudo apt install cmake

```
$ sudo apt install cmake
```

2.3.2 安装 Gcc

命令：sudo apt install build-essential

```
$ sudo apt install build-essential
```

2.4 搭建可选开发环境

2.4.1 安装 Opencv【可显示相机图像】

执行命令：sudo apt-get install libopencv-dev

```
$ sudo apt-get install libopencv-dev
```

2.4.2 安装 Pcl【可显示点云】

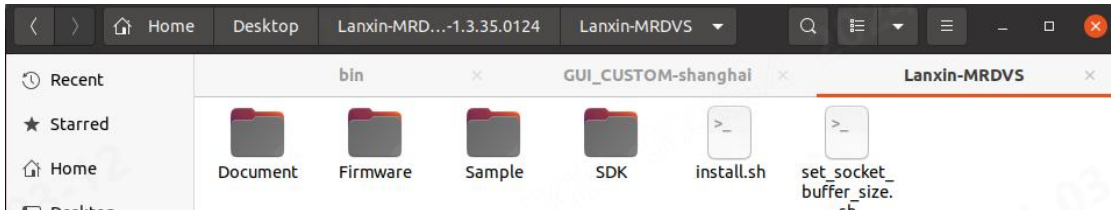
执行命令：sudo apt-get install libpcl-dev

```
$ sudo apt-get install libpcl-dev
```

三、示例代码的介绍与运行准备

3.1 执行安装脚本

- ❖ 解压缩 Lanxin-MRDVS 文件，并进入 Lanxin-MRDVS 文件夹



- ❖ 右键打开控制台
- ❖ 执行 install.sh 脚本

```
/Lanxin-MRDVS$ sudo sh ./install.sh
```

- ❖ 进入/opt/目录检查是否安装成功

```
~/Desktop/Lanxin-MRDVS-1.3.35.0124/Lanxin-MRDVS$ cd /opt/  
/opt$ ll | grep Lanxin  
oot root 4096 Mar 12 21:00 Lanxin-MRDVS/
```

若是能找到文件夹，则是安装成功

若是安装失败，则需要手动添加到 opt 目录下

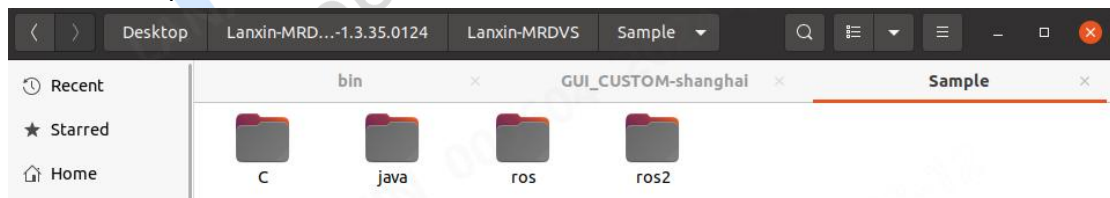
- ❖ 执行 set_socket_buffer_size.sh 脚本

```
$ sudo sh ./set_socket_buffer_size.sh
```

至此安装脚本全部执行完成！

3.2 示例代码介绍

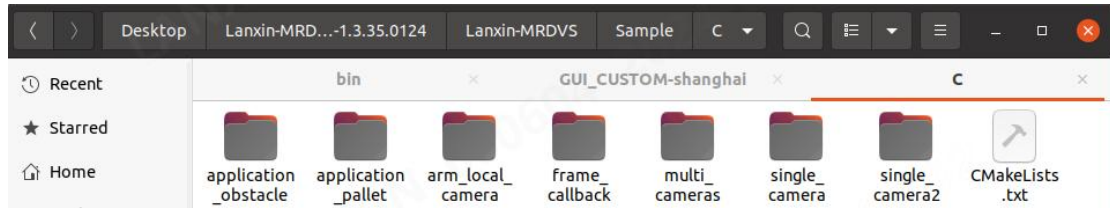
进入到 Sample 目录下，目录内容如下：



- ❖ C 文件夹中包含 C 与 C++相关的代码
- ❖ Java 文件夹中包含一个 http 服务器，用于间接的调用 sdk
- ❖ Ros 文件夹中包含一个 ros 的示例
- ❖ Ros2 文件夹中包含一个 ros2 的示例

3.2.1 C 与 C++示例代码

进入 C 目录，目录内容如下：



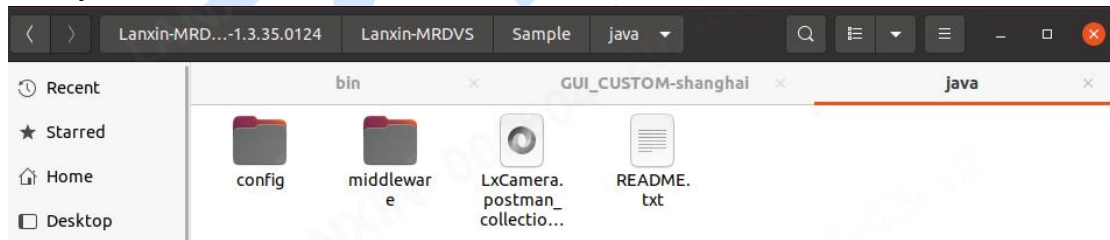
含 9 个文件夹与 1 个 Cmakelist 文件，每个文件夹中包含一个 cpp 文件
每个 cpp 大致实现的功能如下：

- ❖ application_obstacle：获取相机流数据并开启避障算法，输出算法结果
- ❖ application_pallet：获取相机流数据并开启托盘算法，输出算法结果
- ❖ arm_local_camera：在相机中【ARM】调用 SDK 并获取流数据
- ❖ frame_callback：使用回调方式获取相机流数据
- ❖ multi_cameras：打开多个相机并获取流数据
- ❖ single_camera：打开单个相机并获取流数据
- ❖ single_camera2：通过结构体方式获取帧数据

可以根据不同需要选择不同的示例代码进行测试和修改

3.2.2 Java 示例代码

进入 java 目录，目录内容如下：

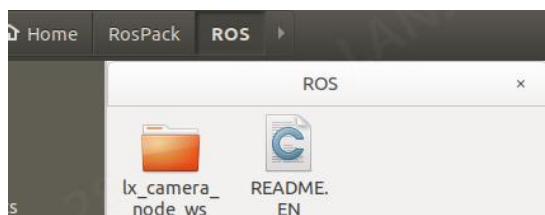


目录中包含 2 个文件夹以及一个 postman.json 文件和 README

- ❖ Config：文件夹中存放着服务器相关的配置
- ❖ Middleware：服务器主题
- ❖ LxCamera.postman_collection.json：postman 配置文件
- ❖ README：java 虚拟服务器使用教程

3.2.3 ROS 示例代码

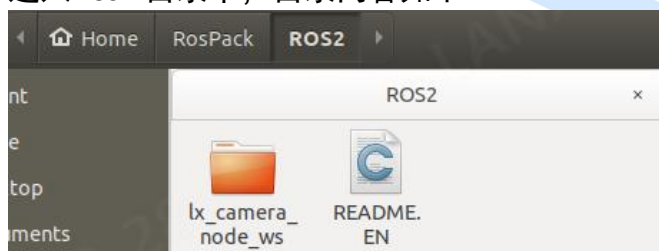
进入 ros 目录中，目录内容如下：



- ❖ lx_camera_node_ws: 文件夹中包含了一个标准的 ros 驱动
- ❖ README.EN: 用英文说明了如何编译节点以及创建一个新的节点

3.2.4 ROS2 示例代码

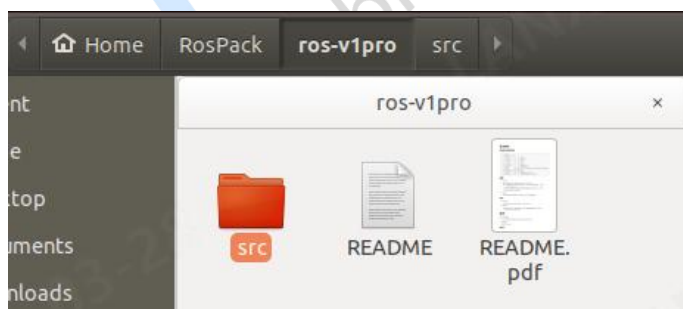
进入 ros2 目录中，目录内容如下：



- ❖ lx_camera_node_ws: 文件夹中包含了一个标准的 ros2 驱动
- ❖ README.EN: 用英文说明了如何编译节点以及创建一个新的节点

3.2.5 ros-v1pro 示例代码

进入 ros-v1pro 目录中，目录内容如下：

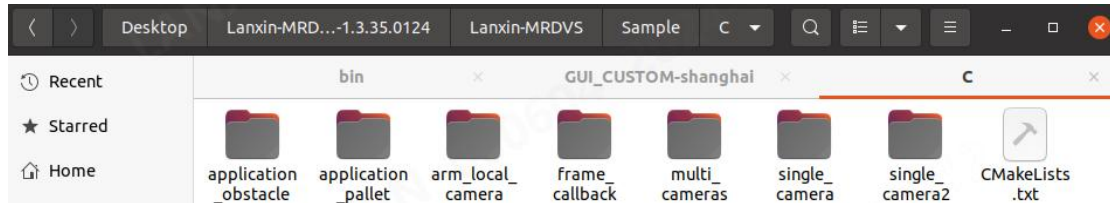


- ❖ src: 文件夹中包含了一个标准的 ros 驱动
- ❖ README: 说明了如何使用这个节点
- ❖ README.pdf: 说明了如何使用这个节点

四、示例代码的编译与运行

4.1 C 与 C++示例代码的编译与运行

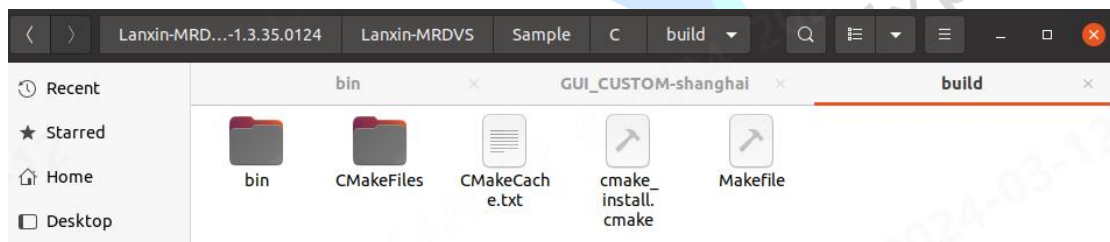
进入示例中的 C 文件夹：



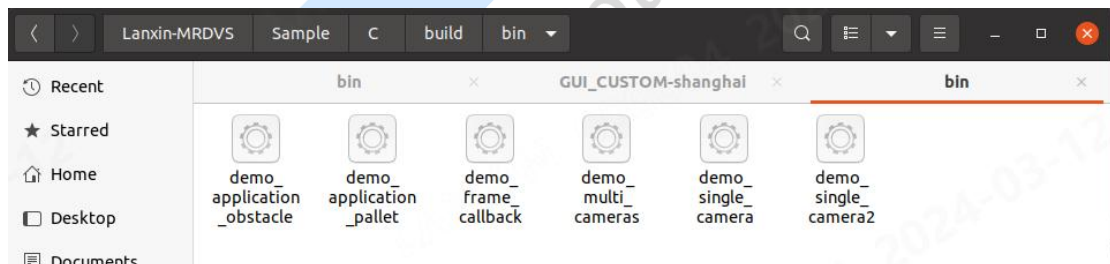
在 C 文件夹中打开一个终端，依次执行以下命令：

```
❖ mkdir build
❖ cd build
❖ cmake ..
❖ make
```

成功执行后进入 build 文件夹中，应当如下：



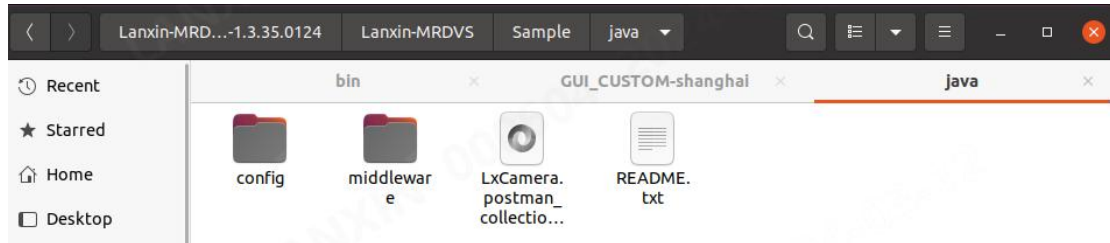
可执行文件就在 bin 目录中，进入 bin 目录应当如下：



除 arm 示例外，其余示例都会编译，可执行文件名与示例文件夹名相同，直接运行即可

4.2 java 示例运行流程

Java 与其他示例代码不相同，它不直接提供源码，它提供一个 http 服务器以及一个 postman 示例，有需要通过 http 来实现间接控制相机的，可以选择 java 示例，进入 java 文件夹，文件夹内容如下：



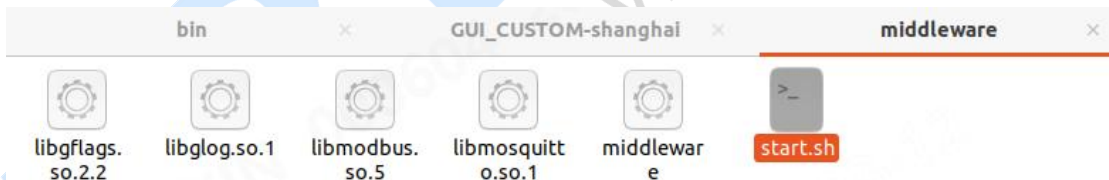
运行流程如下：

- ❖ 进入 config 文件夹中，修改 Configs.json 文件中的 http_sever_host 字段和 http_sever_port 字段：

```
"http_host" : "192.168.1.229",  
"http_port" : 8060,  
"http_sever_host" : "192.168.1.229",  
"http_sever_port" : 8060,  
"mqtt_host" : "192.168.1.229",  
"mqtt_port" : 1884,  
"poll_host" : "192.168.1.227",  
"poll_port" : 502,  
"salve_host" : "192.168.1.229",  
"salve_port" : 1236
```

写本机ip
写未占用端口

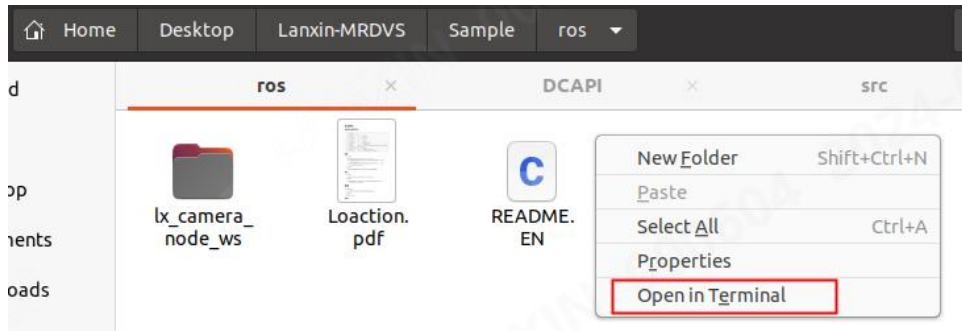
- ❖ 进入 middleware 文件夹，执行 start.sh 脚本，即可启用中转服务器



- ❖ 使用 postman 打开 LxCamera.postman_collection.json 文件，即可向示例端口发送数据

4.3 ROS 示例代码的编译与运行

进入 ROS 文件夹，开启一个控制台，执行：roscore



```
/Desktop/Lanxin-MRDVS/Sample/ros/lx_camera_node_ws$ roscore
```

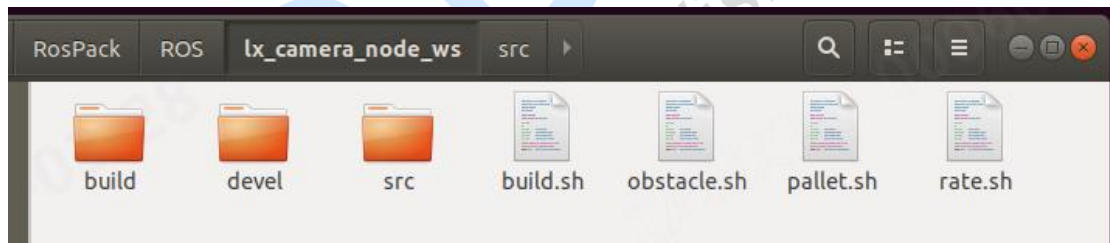
右键再开启一个控制台，进入 lx_camera_node_ws 文件夹执行 catkin_make

```
~/Desktop/Lanxin-MRDVS/Sample/ros$ cd lx_camera_node_ws/  
~/Desktop/Lanxin-MRDVS/Sample/ros/lx_camera_node_ws$ catkin_make
```

成功执行后会显示如下：

```
/usr/include/pcl-1.8/pcl/sample_consensus/model_types.h:99:3: note: declared here  
[ 97%] Linking CXX executable /home/fr1511b/RosPack/ROS/lx_camera_node_ws/devel/lib/lx_camera_ros/location_node  
[ 97%] Built target location_node  
[100%] Linking CXX executable /home/fr1511b/RosPack/ROS/lx_camera_node_ws/devel/lib/lx_camera_ros/lx_camera_node  
[100%] Built target lx_camera_node  
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$
```

此时文件夹内 devel 和 build 是编译生成的：



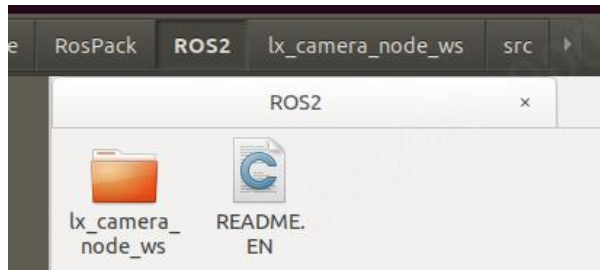
执行 source devel/setup.bash 配置环境后使用 roslaunch 运行节点，命令如下：

- ❖ roslaunch lx_camera_ros lx_camera_ros.launch
- ❖ roslaunch lx_camera_ros obstacleV2.launch
- ❖ roslaunch lx_camera_ros obstacle.launch
- ❖ roslaunch lx_camera_ros pallet.launch

从上往下依次是：运行普通节点、运行避障算法 2 节点、运行避障算法节点、运行托盘算法节点，各节点不能同时启用。

4.4 ROS2 示例代码的编译与运行

进入 ros2 文件夹，文件夹内容如下：



示例 ros 节点采用 colcon 编译方式

首先需要安装编译工具，执行：`sudo apt install python3-colcon-common-extensions`

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

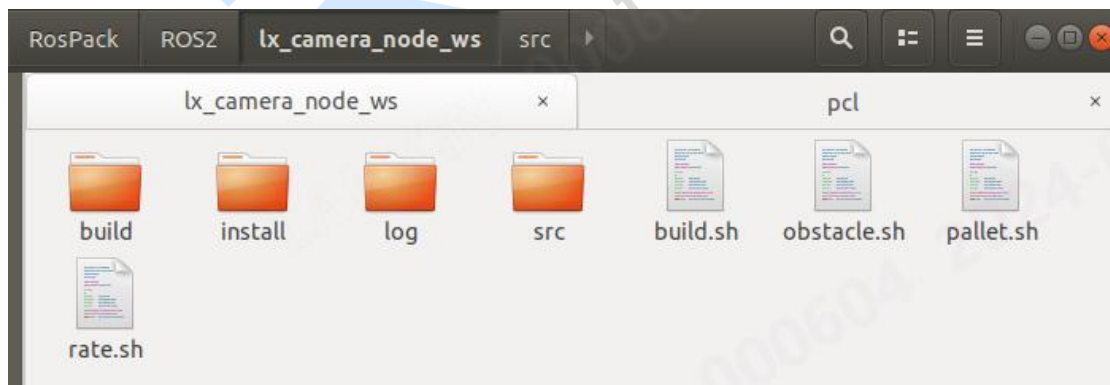
然后安装 ros-pcl 依赖包：`sudo apt-get install ros-foxy-pcl*`【红色字体部分需要替换为对应的 ros2 版本】

安装完成后进入 lx_camera_node_ws 目录编译项目，执行：`colcon build`

根据电脑性能不同，编译通常会需要 30 秒到 120 秒，成功编译后会显示如下：

```
---
Finished <<< lx_camera_ros [1min 44s]
Summary: 1 package finished [1min 45s]
1 package had stderr output: lx_camera_ros
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$
```

部分警告是因为版本兼容问题，无需处理，编译完成后文件夹内容如下：



执行 `source install/setup.bash` 配置环境后使用 `ros2 launch` 运行节点，命令如下：

- ❖ `ros2 launch lx_camera_ros lx_camera_ros_launch.py`
- ❖ `ros2 launch lx_camera_ros obstacleV2.launch.py`
- ❖ `ros2 launch lx_camera_ros obstacle.launch.py`
- ❖ `ros2 launch lx_camera_ros pallet.launch.py`

从上往下依次是：运行普通节点、运行避障算法 2 节点、运行避障算法节点、运行托盘算法节点，各节点不能同时启用。

4.5 ros-v1pro 示例代码的编译与运行

ros-v1pro 是 V1PRO 相机专用 ros 驱动，在 ros-v1pro 文件夹中有 README.pdf 对该驱动详细说明，本文档中只对其做简单介绍：



进入 ros-v1pro 文件夹中右键打开控制台，使用 `catkin_make` 编译，编译完成后显示如下：

```
[100%] Linking CXX executable /home/fr1511b/RosPack/ros-v1pro/devel/lib/lx_camera_ros/lx_camera_node
[100%] Built target lx_camera_node
fr1511b@ubuntu:~/RosPack/ros-v1pro$
```

进行环境配置：`source devel/setup.bash`

```
[100%] Linking CXX executable lx_camera_node
[100%] Built target lx_camera_node
fr1511b@ubuntu:~/RosPack/ros-v1pro$ source devel/setup.bash
fr1511b@ubuntu:~/RosPack/ros-v1pro$
```

最后运行节点：

- ❖ `roslaunch lx_camera_ros localization.launch`
- ❖ `roslaunch lx_camera_ros mapping.launch`
- ❖ `roslaunch lx_camera_ros sensor_sim.launch`

从上往下依次是：运行定位节点，运行建图节点，运行虚拟数据节点，各节点不能同时启用。

五、示例代码解释与说明

在这个部分中，所有代码与伪代码将会分成几个部分来讲，一般来说一个小节内的都属于同一个文件，除特殊文件做详细解释外，源码文件只解释大致模块，详细的解释在代码中有注释。

5.1 C 与 C++

5.1.1 CMakeLists

```
cmake_minimum_required(VERSION 3.0) 指定最低的cmake 版本
project(lx_camera_demo) 设置在工程中的项目名

set(LIBS LxCameraApi) 定义变量，值为LxCameraApi

if (WIN32) 如果是windows
    if(CMAKE_CL_64) 如果是64位
        set(PLATFORM_PREFEX win_x64) 定义变量，值为win_x64
    else() 否则
        set(PLATFORM_PREFEX win_x86) 定义变量，值为win_x86
    endif() if的结束标志
    set(CAMERA_BASE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/../..) 定义变量，值为路径
    include_directories(${CAMERA_BASE_PATH}/SDK/include) 设置包含目录，目录为SDK头文件目录
    link_directories(${CAMERA_BASE_PATH}/SDK/lib/${PLATFORM_PREFEX}) 设置链接库目录
elseif ([UNIX]) 如果是linux
    include_directories(/opt/Lanxin-MRDVS/include) 设置包含目录，目录为opt下SDK头文件目录
    link_directories(/opt/Lanxin-MRDVS/lib/) 设置链接库目录
    list(APPEND LIBS ${LIBS} pthread) 在LIBS变量中添加pthread
endif () if的结束标志
set(EXECUTABLE_OUTPUT_PATH ${CMAKE_CURRENT_BINARY_DIR}/bin) 设置二进制文件的输出目录
```

```
#opencv 注释
find_package(OpenCV) 查找OpenCV依赖包
if(${OpenCV_FOUND}) 如果找到了
    message("find opencv " ${OpenCV_VERSION}) 在控制台打印opencv版本
    set(CMAKE_CXX_STANDARD 11) 设置要使用的Cmake 版本
    set(CMAKE_CXX_STANDARD_REQUIRED TRUE) 设置强制使用指定的cmake 版本
    include_directories(${OpenCV_INCLUDE_DIRS}) 设置包含目录
    link_directories(${OpenCV_LIB_DIR}) 设置链接库目录
    list(APPEND LIBS ${LIBS} ${OpenCV_LIBS}) 在LIBS变量中添加opencv库
    add_definitions(-DHAS_OPENCV) 添加编译宏定义
endif() if的结束标志
```

```
macro(add_exe NAME) 定义一个宏，宏名为add_exe，参数为：NAME
add_executable(demo_${NAME} ${NAME}/${NAME}.cpp) 在工程中添加可执行项目和对应cpp
target_link_libraries(demo_${NAME} ${LIBS}) 将链接库添加到目标文件
endmacro() 宏定义结束标志

add_exe(single_camera) 引用宏，参数为single_camera，以下相同，只是参数变化，在这里相当于在工程添加了6个可执行文件的项目，编译后也会产生6个可执行文件
add_exe(single_camera2)
add_exe(multi_cameras)
add_exe(application_obstacle)
add_exe(application_pallet)
add_exe(frame_callback)
```

5.1.2 application_obstacle 和 application_pallet

```
#include <iostream>
#include <sstream>          添加包含头文件
#include <thread>
#include <string.h>
#include "lx_camera_api.h"
#include "lx_camera_application.h" 因为包含算法，所以也需要包含算法的.h头文件
using namespace std;

#define checkTC(state) {LX_STATE val=state; 用于检查SDK api 的返回值是否正确 \--
}

int PrintData(void* data_ptr); 声明函数

int main(int argc, char** argv) main函数，包含查找相机、打开相机、设置与读取相机参数等操作
{
}

int PrintData(void* data_ptr) 打印算法输出到控制台
{
}
```

5.1.3 arm_local_camera

```
#include <iostream>
#include <thread>
#include <sstream>          包含头文件
#include "lx_camera_api.h"

using namespace std;

DCHandle handle = 0; 定义sdk相机句柄
#define checkTC(state) {LX_STATE val=state; 检查SDK api 返回值 \--
}

int TestDepth(bool is_enable, DCHandle handle);
int TestAmp(bool is_enable, DCHandle handle); 申明函数
int TestRgb(bool is_enable, DCHandle handle);

int main(int argc, char** argv) main函数，包含查找相机、打开相机、启流、调用函数、设置获取参数等
{
}

int TestDepth(bool is_enable, DCHandle handle) 获取深度流数据，并获取深度值，并获取点云
{
}

int TestAmp(bool is_enable, DCHandle handle) 获取强度流数据，并获取强度值
{
}

int TestRgb(bool is_enable, DCHandle handle) 获取RGB流数据，并获取rgb值
{
}
```

5.1.4 frame_callback

```
#include <string>
#include "lx_camera_api.h"           包含头文件
#include "lx_camera_application.h"  因为回调结果中包含算法结果，所以也要包含算法头文件
#ifdef HAS_OPENCV 判断是否有opencv宏定义
#include "opencv2/opencv.hpp"  如果存在opencv宏定义则包含opencv头文件
using namespace cv;  如果存在opencv宏定义则申明命名空间
#endif 结束ifdef标志

static char wait_key = '0';
#define checkTC(state) {LX_STATE val=state; 函数用于检查SDK api 接口返回值是否正确 \ ...
}

void CallbackFunc(FrameInfo* frame, void* usr_data); 申明回调函数

int main(int argc, char** argv) main函数，同之前相似，不过多了一个回调函数的注册
{ ...
}

void CallbackFunc(FrameInfo* frame_ptr, void* usr_data) 回调函数主体，SDK 获取到新帧后会调用这个函数
{ ...
#ifdef HAS_OPENCV ...           如果存在opencv宏定义则会显示图像，否则不显示
#endif ...
}
```

5.1.5 multi_cameras

```
#include <iostream>
#include <string>
#include <sstream>
#include <thread>           包含头文件
#include <string.h>
#include <vector>
#include "lx_camera_api.h"
#ifdef HAS_OPENCV
#include "opencv2/opencv.hpp"  通过宏定义判断是否添加opencv
using namespace cv;
#endif
using namespace std;

static char wait_key = '0';
#define checkTC(state, handle) {LX_STATE val=state; 检查SDK api的返回值 \ ...
}

int thread_work(std::string ip); 申明线程函数

int main(int argc, char** argv) main函数，与之前相同，只是多了一个线程的注册操作
{ ...
}

int thread_work(std::string ip)
{ ...
#ifdef HAS_OPENCV ...           每查找到一个相机，就会开启一个线程，函数中获取流数据并显示
#endif ...
#ifdef HAS_OPENCV ...
#endif ...
}
```

5.1.6 single_camera

```

#include <iostream>
#include <thread>
#include <sstream>
#include "lx_camera_api.h"

#ifdef HAS_OPENCV
#include "opencv2/opencv.hpp"
using namespace cv;
#endif
using namespace std;

static char wait_key = '0';
static DcHandle handle = 0;

static int tof_width = 0;
static int tof_height = 0;
static int tof_depth_type = LX_DATA_TYPE::LX_DATA_UNSIGNED_SHORT;
static int tof_amp_type = LX_DATA_TYPE::LX_DATA_UNSIGNED_SHORT;
static int rgb_width = 0;
static int rgb_height = 0;
static int rgb_channels = 3;
static int rgb_data_type = LX_DATA_TYPE::LX_DATA_UNSIGNED_CHAR;

#define checkTC(state) {LX_STATE val=state; 检查SDK api 返回值 \ ...
}

int TestDepth(bool is_enable, DcHandle handle);
int TestAmp(bool is_enable, DcHandle handle);
int TestRgb(bool is_enable, DcHandle handle);
void WaitKey()
{ ...
}

int main(int argc, char** argv)
{ ...
}

int TestDepth(bool is_enable, DcHandle handle)
{ ...
#ifdef HAS_OPENCV ...
#endif ...
}

int TestAmp(bool is_enable, DcHandle handle)
{ ...
#ifdef HAS_OPENCV ...
#endif ...
}

int TestRgb(bool is_enable, DcHandle handle)
{ ...
#ifdef HAS_OPENCV ...
#endif ...
}

```

5.1.7 single_camera2

```
#include <string>
#include <thread>
#include "lx_camera_api.h"

#ifdef HAS_OPENCV
#include "opencv2/opencv.hpp"
using namespace cv;
#endif
using namespace std;

static char wait_key = '0';
#define checkTC(state) {LX_STATE val=state; 检查SDK api 返回值是否正确 \ ...
}

int TestFrame(DcHandle handle);    申明函数
void WaitKey()
{
    ... 定义WaitKey函数
}

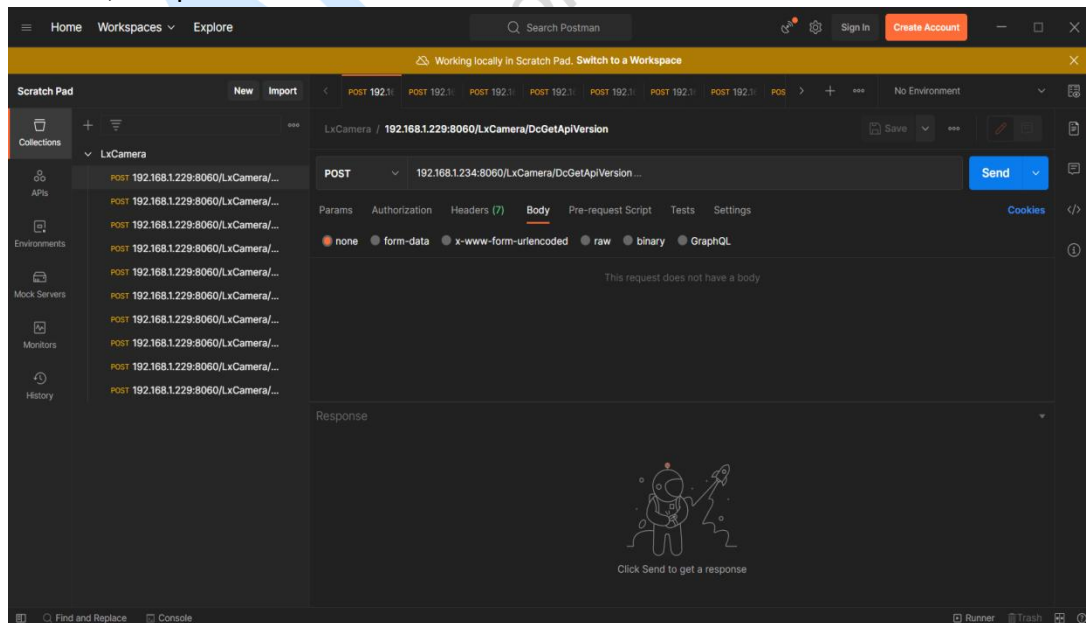
int main(int argc, char** argv)
{
    ... main函数
}

int TestFrame(DcHandle handle)
{
    ...
#ifdef HAS_OPENCV ...
#endif
#ifdef HAS_OPENCV ...
#endif
#ifdef HAS_OPENCV ...
#endif
}

// 获取帧数据函数，此函数中获取的帧在一个结构体中，读取结构体中的帧信息并显示
```

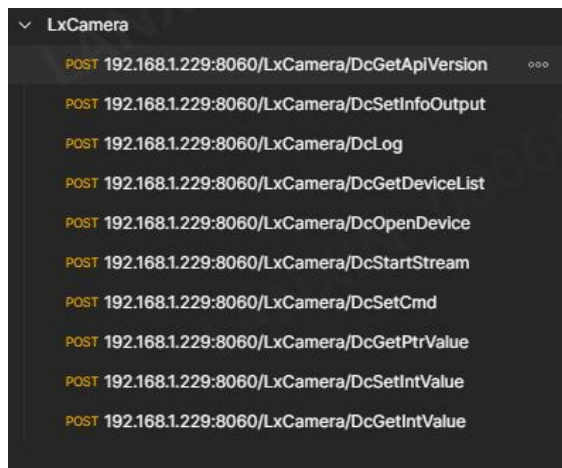
5.2 java

Java 示例中提供的仅是一个 HTTP 服务器，因此说明也只是对接口以及消息格式的说明，用 postman 打开示例文件后如下图：



消息都是 post 方式，在发送前，记得修改 IP 为服务器的 ip，但是 url 不要修改，url 为服务器内部写死，响应的消息格式与内容都在 body 中

示例中提供的接口如下：



- ❖ DcGetApiVersion: 获取 SDK 的版本号吗，没有参数输入
- ❖ DcSetInfoOutput: 设置 SDK 日志输出的路径，参数如下：

```
1 {
2   "log_path" : "./",
3   "print_level" : 2,
4   "enable_screen_print" : 0
5 }
```

- ✧ log_path: 日志输出的路径
- ✧ print_level: 日志打印的等级
- ✧ enable_screen_print: 日志是否在控制台打印
- ❖ DcLog: 输出日志到 SDK 日志中，任何输入的内容都会被打印到 SDK 日志中
- ❖ DcGetDeviceList: 获取设备列表，没有输入，输入为 JSON
- ❖ DcOpenDevice: 打开设备，输入如下：

```
{
  "open_mode" : 0,
  "param" : "0"
}
```

- ✧ open_mode: 打开相机的模式
- ✧ param: 打开相机的参数
- ❖ DcStartStream: 让某一个相机开始推送流数据，输入参数如下：

```
{
  "handle" : 174539057817095
}
```

- ✧ handle: 相机的句柄，在 DcOpenDevice 中会有返回
- ❖ DcSetCmd: 对相机下发命令，输入如下：

```
{
  "handle" : 174539057817095,
  "cmd" : 5001
}
```

- ✧ handle: 要设置的相机的句柄
- ✧ cmd: 指令编号
- ❖ DcGetPtrValue: 获取指针数据 (图像, 参数等), 输入如下:
 - ✧ handle: 要设置的相机的句柄
 - ✧ cmd: 指令编号
- ❖ DcSetIntValue: 设置相机 int 型的参数, 输入如下:

```
{
  "handle" : 174539057817095,
  "cmd" : 1001,
  "value" : 100
}
```

- ✧ handle: 要设置的相机的句柄
- ✧ cmd: 指令编号
- ✧ value: 要设置的值
- ❖ DcGetIntValue: 获取设备 int 型的参数, 输入如下:

```
{
  "handle" : 174539057817095,
  "cmd" : 1001
}
```

- ✧ handle: 要设置的相机的句柄
- ✧ cmd: 指令编号
- ❖ 其他相关说明参见下表:

发	{ "log_path" : "./", "print_level" : 2, "enable_screen_print" : 0 }	收	{ "errCode" : 0 }
发	{ "open_mode" : 0, "param" : "0" }	收	{ "algor_ver" : "VER_1.0.2.230315", "dev_type" : "M4 camera", "errCode" : 0, "firmware_ver" : "V1.0.0.3_230315", "handle" : 174539057817095, "id" : "f13141144bb1", "ip" : "192.168.100.16:3956", "mac" : "9e:be:0a:8a:66:07", "name" : "camera_M4_192.168.100.16_3956", "sn" : "9a5ed62d22b34bb1" }
发	{ "handle" : 174539057817095, "cmd" : 6001 }	收	{ "errCode" : 0, "value" : "\u0011\u0001\u0014\u0001..." }

5.3 ROS1 & ROS2

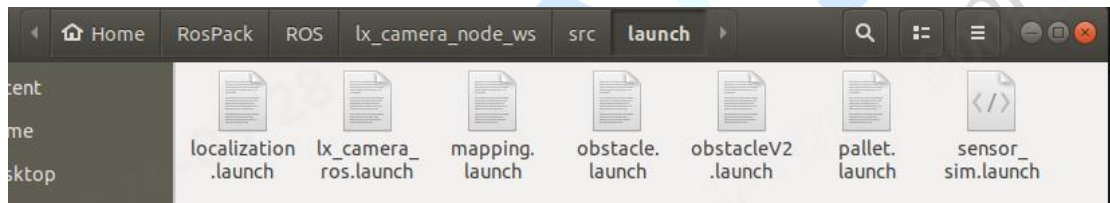
在 ros 说明中，不会讲述驱动源码相关内容，而是讲述 launch 文件中的配置所对应的内容，如何调整配置，如何发送以及订阅消息。提供的 ros 节点中已经包含了大部分功能，所以不建议直接修改驱动源码。如有特殊需要可以先行了解 C++ 示例中的代码流程后再对驱动源码进行修改。

执行 ros 驱动时，必须确保系统默认路径中 opencv 和 pcl 的版本唯一，否则 ros 链接库时会发生错误，可能导致编译失败或者运行崩溃。

5.3.1 ROS1 launch 文件说明

5.3.1.1 所有的 launch 文件

在 src/launch 文件夹中有如下文件：



- ❖ localization.launch：视觉定位节点的运行文件
- ❖ lx_camera_ros.launch：普通节点的运行文件
- ❖ mapping.launch：视觉定位建图的运行文件
- ❖ obstacle.launch：避障算法的运行文件
- ❖ obstacleV2.launch：避障算法 V2 的运行文件
- ❖ pallet.launch：托盘算法的运行文件
- ❖ sensor_sim.launch：模拟定位数据的运行文件

可以通过运行不同的 launch 文件来执行节点的不同功能，所有节点中除 localization、mapping 和 sensor 之外，参数内容都是一致的，只是配置有所不同，在以下小节中会详细介绍，对于 localization、mapping 和 sensor 三个 launch 文件的参数详细介绍请看 ros-v1pro 节点中的介绍。

5.3.1.2 相机 ip、流配置、工作模式配置、点云单位等

```
<!-- ip、日志路径、流配置、算法、工作模式配置、点云单位 -->
<param name="ip" value="0" />
<param name="log_path" value="/var/log/" />
<param name="is_xyz" value="1" />
<param name="is_depth" value="1" />
<param name="is_amp" value="1" />
<param name="is_rgb" value="1" />
<param name="lx_work_mode" value="0" />
<param name="lx_application" value="0" />
<param name="lx_tof_unit" value="1" />
```

- ❖ ip: 需要连接的相机 IP, 若为 0 则默认连接第一个
- ❖ log_path: SDK 日志储存路径, 需要填入绝对路径
- ❖ is_xyz: 是否推送点云数据
- ❖ is_depth: 是否推送深度数据
- ❖ is_amp: 是否推送强度图数据
- ❖ is_rgb: 是否推送 RGB 图数据
- ❖ lx_work_mode: 工作模式
- ❖ lx_application: 是否开启算法, 0: 关闭, 1: 避障算法, 2: 托盘算法, 3: 定位算法, 4: 避障算法 V2
- ❖ lx_tof_unit: 点云单位, 1 为米, 0 为毫米, 使用 rviz 查看时需要修改为 1

5.3.1.3 相机位姿配置

```
<!-- 相机位姿配置 -->
<param name="x" value="0" />
<param name="y" value="0" />
<param name="z" value="0" />
<param name="yaw" value="90" />
<param name="roll" value="90" />
<param name="pitch" value="90" />
```

可以配置点云的旋转角度以及平移向量。

- ❖ X: 相机距离点云中心的 X 平移量
- ❖ Y: 相机距离点云中心的 Y 平移量
- ❖ Z: 相机距离点云中心的 Z 平移量
- ❖ Yaw: 点云绕 Z 轴旋转的角度
- ❖ Roll: 点云绕 X 轴旋转的角度
- ❖ Pitch: 点云绕 Y 轴旋转的角度

5.3.1.4 是否使用 launch 配置

```
<!-- 是否使用launch配置 -->
<param name="raw_param" value="0" />
```

可以选择是否使用 launch 文件中的相机参数配置。

❖ raw_param: 0: 使用相机内的配置, 1: 使用 launch 文件中的配置
不论此参数设置为 0 还是 1, 在此参数之前的所有参数都会被设置, 0 和 1 只对此参数之后的参数有效。

5.3.1.5 2D 参数配置

```
<!-- 2D配置 -->
<param name="lx_2d_binning" value="0" />
<param name="lx_2d_undistort" value="0" />
<param name="lx_2d_undistort_scale" value="51" />
<param name="lx_2d_auto_exposure" value="0" />
<param name="lx_2d_auto_exposure_value" value="11" />
<param name="lx_2d_exposure" value="10001" />
<param name="lx_2d_gain" value="101" />
```

- ❖ lx_2d_binning: 0: 1x1, 1: 2x2, 2: 4x4
- ❖ lx_2d_undistort: 2d 反畸变, 0: 关闭, 1: 开启
- ❖ lx_2d_undistort_scale: 2D 反畸变系数
- ❖ lx_2d_auto_exposure: 2D 自动曝光, 0: 关闭, 1: 开启
- ❖ lx_2d_auto_exposure_value: 2D 自动曝光目标强度
- ❖ lx_2d_exposure: 2D 曝光值
- ❖ lx_2d_gain: 2D 增益

5.3.1.6 3D 参数配置

```
<!-- 3D配置 -->
<param name="lx_rgb_to_tof" value="0" />
<param name="lx_3d_binning" value="0" />
<param name="lx_multit_mode" value="0" />
<param name="lx_3d_undistort" value="0" />
<param name="lx_3d_undistort_scale" value="0" />
<param name="lx_hdr" value="0" />
<param name="lx_3d_auto_exposure" value="1" />
<param name="lx_3d_auto_exposure_value" value="50" />
<param name="lx_3d_first_exposure" value="1100" />
<param name="lx_3d_second_exposure" value="200" />
<param name="lx_3d_gain" value="11" />
```

- ❖ lx_rgb_to_tof: RGB 对齐
- ❖ lx_3d_binning: 0: 1x1, 1: 2x2, 2: 4x4
- ❖ lx_multit_mode: 多机模式, 可以排除多机干扰, 0: 关闭, 1: 开启
- ❖ lx_3d_undistort: 3d 反畸变, 0: 关闭, 1: 开启
- ❖ lx_3d_undistort_scale: 3D 反畸变系数
- ❖ lx_hdr: hdr, 0: 关闭, 1: 开启
- ❖ lx_3d_auto_exposure: 3D 自动曝光, 0: 关闭, 1: 开启
- ❖ lx_3d_auto_exposure_value: 3D 自动曝光目标强度
- ❖ lx_3d_first_exposure: 3D 高积分
- ❖ lx_3d_second_exposure: 3D 低积分
- ❖ lx_3d_gain: 3D 增益

5.3.1.7 最大最小深度

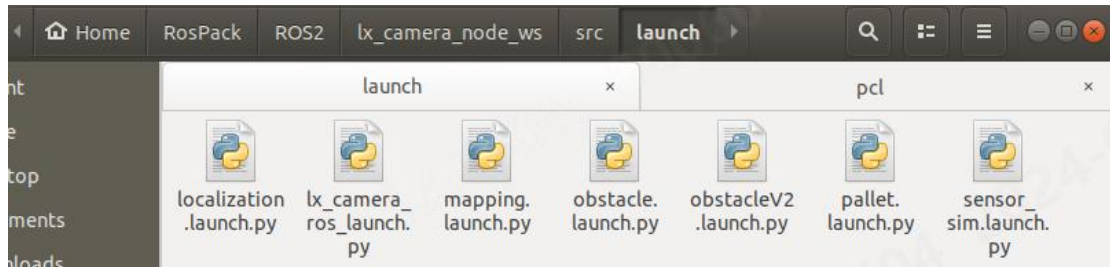
```
<!-- 深度 -->
<param name="lx_min_depth" value="0" />
<param name="lx_max_depth" value="8000" />
```

- ❖ lx_min_depth: 最小深度
- ❖ lx_max_depth: 最大深度

5.3.2 ROS2 launch 文件说明

5.3.2.1 所有的 launch 文件

在 src/launch 文件夹中有如下文件：



- ❖ localization.launch.py: 视觉定位节点的运行文件
- ❖ lx_camera_ros.launch.py: 普通节点的运行文件
- ❖ mapping.launch.py: 视觉定位建图的运行文件
- ❖ obstacle.launch.py: 避障算法的运行文件
- ❖ obstacleV2.launch.py: 避障算法 V2 的运行文件
- ❖ pallet.launch.py: 托盘算法的运行文件
- ❖ sensor_sim.launch.py: 模拟定位数据的运行文件

可以通过运行不同的 launch 文件来执行节点的不同功能，所有节点中除 **localization**、**mapping** 和 **sensor** 之外，参数内容都是一致的，只是配置有所不同，在以下小节中会详细介绍，对于 localization、mapping 和 sensor 三个 launch 文件的参数详细介绍请看 ros-v1pro 节点中的介绍。

5.3.2.2 相机 ip、流配置、工作模式配置、点云单位等

```
#<!-- ip、日志路径、流配置、算法、工作模式配置、点云单位 -->
{"ip" : "0"},
{"log_path" : "/var/log/"},
{"is_xyz" : 1 },
{"is_depth" : 1 },
{"is_amp" : 1 },
{"is_rgb" : 1 },
{"lx_work_mode" : 0},
{"lx_application" : 0},
{"lx_tof_unit" : 1},
```

- ❖ ip: 需要连接的相机 IP, 若为 0 则默认连接第一个
- ❖ log_path: SDK 日志储存路径, 需要填入绝对路径
- ❖ is_xyz: 是否推送点云数据
- ❖ is_depth: 是否推送深度数据
- ❖ is_amp: 是否推送强度图数据
- ❖ is_rgb: 是否推送 RGB 图数据
- ❖ lx_work_mode: 工作模式
- ❖ lx_application: 是否开启算法, 0: 关闭, 1: 避障算法, 2: 托盘算法, 3: 定位算法, 4: 避障算法 V2
- ❖ lx_tof_unit: 点云单位, 1 为米, 0 为毫米, 使用 rviz 查看时需要修改为 1

5.3.2.3 相机位姿配置

```
#<!-- 相机位姿配置 -->
{"x" : 0},
{"y" : 0},
{"z" : 0},
{"yaw" : 0},
{"roll" : 0},
{"pitch" : 90},
```

可以配置点云的旋转角度以及平移向量。

- ❖ X: 相机距离点云中心的 X 平移量
- ❖ Y: 相机距离点云中心的 Y 平移量
- ❖ Z: 相机距离点云中心的 Z 平移量
- ❖ Yaw: 点云绕 Z 轴旋转的角度
- ❖ Roll: 点云绕 X 轴旋转的角度
- ❖ Pitch: 点云绕 Y 轴旋转的角度

5.3.2.4 是否使用 launch 配置

```
#<!-- 是否使用launch配置 -->
{"raw_param": 0},
```

可以选择是否使用 launch 文件中的相机参数配置。

❖ raw_param: 0: 使用相机内的配置, 1: 使用 launch 文件中的配置
不论此参数设置为 0 还是 1, 在此参数之前的所有参数都会被设置, 0 和 1 只对此参数之后的参数有效。

5.3.2.5 2D 参数配置

```
#<!-- 2D配置 -->
{"lx_2d_binning" : 0 },
{"lx_2d_undistort" : 0 },
{"lx_2d_undistort_scale" : 51 },
{"lx_2d_auto_exposure" : 0 },
{"lx_2d_auto_exposure_value" : 11 },
{"lx_2d_exposure" : 10001},
{"lx_2d_gain" : 101 },
```

- ❖ lx_2d_binning: 0: 1x1, 1: 2x2, 2: 4x4
- ❖ lx_2d_undistort: 2d 反畸变, 0: 关闭, 1: 开启
- ❖ lx_2d_undistort_scale: 2D 反畸变系数
- ❖ lx_2d_auto_exposure: 2D 自动曝光, 0: 关闭, 1: 开启
- ❖ lx_2d_auto_exposure_value: 2D 自动曝光目标强度
- ❖ lx_2d_exposure: 2D 曝光值
- ❖ lx_2d_gain: 2D 增益

5.3.2.6 3D 参数配置

```
#<!-- 3D配置 -->
{"lx_rgb_to_tof" : 0 },
{"lx_3d_binning" : 0},
{"lx_multit_mode" : 0 },
{"lx_3d_undistort" : 0},
{"lx_3d_undistort_scale" : 0},
{"lx_hdr" : 0},
{"lx_3d_auto_exposure" : 1},
{"lx_3d_auto_exposure_value" : 50},
{"lx_3d_first_exposure" : 1100},
{"lx_3d_second_exposure" : 200},
{"lx_3d_gain" : 11},
```

- ❖ lx_rgb_to_tof: RGB 对齐
- ❖ lx_3d_binning: 0: 1x1, 1: 2x2, 2: 4x4
- ❖ lx_multit_mode: 多机模式, 可以排除多机干扰, 0: 关闭, 1: 开启
- ❖ lx_3d_undistort: 3d 反畸变, 0: 关闭, 1: 开启
- ❖ lx_3d_undistort_scale: 3D 反畸变系数
- ❖ lx_hdr: hdr,0: 关闭, 1: 开启
- ❖ lx_3d_auto_exposure: 3D 自动曝光, 0: 关闭, 1: 开启
- ❖ lx_3d_auto_exposure_value: 3D 自动曝光目标强度
- ❖ lx_3d_first_exposure: 3D 高积分
- ❖ lx_3d_second_exposure: 3D 低积分
- ❖ lx_3d_gain: 3D 增益

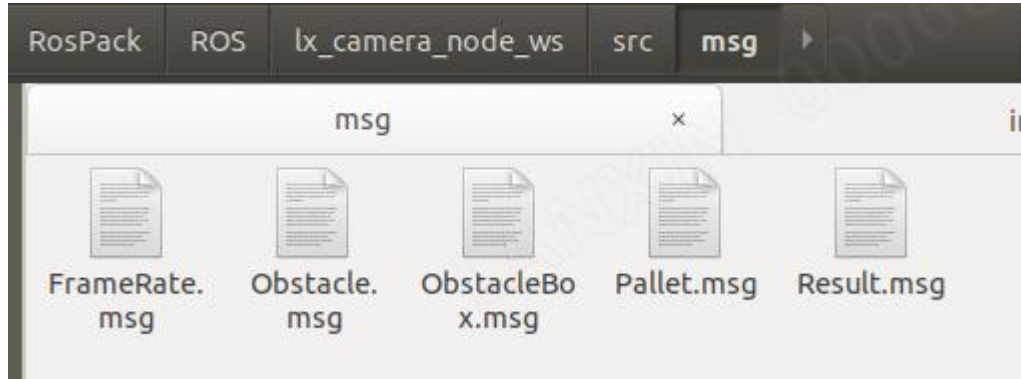
5.3.2.7 最大最小深度

```
#<!-- 深度 -->
{"lx_min_depth" : 0},
{"lx_max_depth" : 8000}
```

- ❖ lx_min_depth: 最小深度
- ❖ lx_max_depth: 最大深度

5.3.3 特有主题说明

因 ROS1 和 ROS2 特有主题类型一致，所以此处 ROS1 为例。
在 src/msg 文件夹中有如下文件：



5.3.3.1 FrameRate.msg

这个节点输出了帧率信息，消息格式如下：

```
Header header
float32 depth
float32 amp
float32 rgb
float32 temperature
```

- ❖ header: 当前数据的帧信息以及时间戳
- ❖ depth: 当深度图开启时，输出深度图帧率
- ❖ amp: 当强度图开启时，输出强度图帧率
- ❖ rgb: 当 rgb 开启时，输出 rgb 帧率
- ❖ temperature: 输出相机温度

节点开始运行后（除视觉定位相关的节点以外），可以订阅 LxCamera_FrameRate 节点获取相关信息，输出如下：

```
---
header:
  seq: 347
  stamp:
    secs: 1711598195
    nsecs: 230031221
  frame_id: "mrdvs"
depth: 24.0
amp: 24.0
rgb: 24.0
temperature: 36.4375
---
```

5.3.3.2 Obstacle.msg

obstacle 输出避障算法结果相关的信息，obstacle 消息格式如下：

```
Header header
int8 status
int32 io_output
int32 box_number
ObstacleBox[] box
```

- ❖ header: 当前数据的帧信息以及时间戳
- ❖ status: 避障返回的状态信息
 - 0: 正常避障
 - 1: 未设置地面
 - 2: 未设置输入
 - 3: 范围内无点云
 - 4: 过滤地面后点云为空
 - 5: 半径滤波后点云为空
 - 6: 未得到分割结果
 - 99: 未定义的错误
- ❖ io_output: io 接口输出的避障信息
 - 0: 正常
 - 1: 警告
 - 2: 停障
- ❖ box_number: 障碍物数量
- ❖ box: 障碍物信息，变长数组，长度由 box_number 决定

节点开始运行后（需开启避障算法），可以订阅 LxCamera_Obstacle 节点获取相关信息，输出如下（图中数据为模拟数据）：

```
---
header:
  seq: 64
  stamp:
    secs: 1711634242
    nsecs: 144264767
  frame_id: "mrdvs"
status: 0
io_output: 1
box_number: 3
box:
-
  center: [366.0, 2743.0, 326.0]
  rotation: [50.0, 958.0, 298.0, 837.0, 950.0, 498.0, 229.0, 969.0, 398.0]
  translation: [804.0, 512.0, 576.0]
  width: 1271.0
  height: 862.0
  depth: 2.0
-
  center: [2255.0, 2349.0, 1108.0]
  rotation: [163.0, 736.0, 764.0, 677.0, 151.0, 415.0, 709.0, 854.0, 114.0]
  translation: [1612.0, 1220.0, 210.0]
  width: 528.0
  height: 918.0
  depth: 2.0
-
  center: [2941.0, 107.0, 471.0]
  rotation: [439.0, 689.0, 440.0, 189.0, 493.0, 304.0, 117.0, 653.0, 151.0]
  translation: [347.0, 908.0, 852.0]
  width: 690.0
  height: 448.0
  depth: 0.0
---
```

5.3.3.3 Obstacle_box.msg

obstacle_box 输出障碍物相关的信息，obstacle_box 消息格式如下：

```
float32[3] center
float32[9] rotation
float32[3] translation
float32 width
float32 height
float32 depth
```

- ❖ center: 障碍物在点云中的质心
- ❖ rotation: 障碍物点云的旋转矩阵
- ❖ translation: 障碍物点云的平移向量
- ❖ width: 障碍物框的宽
- ❖ height: 障碍物框的高
- ❖ depth: 障碍物框的深度

开始运行后（需开启避障算法），可以订阅 LxCamera_Obstacle 节点获取相关信息，输出如下：

```
center: [2.799999952316284, 2.869999885559082, 2.5399999618530273]
rotation: [52.540000915527344, 41.099998474121094, 67.13999938964844, 60.779998779296875, 86.63999938964844, 70.13999938964844, 65.5, 14.529999732971191, 87.9800033569336]
translation: [54.560001373291016, 82.5999984741211, 10.75]
width: 479.0
height: 1028.0
depth: 3763.0
```

5.3.3.4 Result.msg

向节点中 servic 发送指令消息后得到的返回值就储存在 Result 中：

```
int32 ret
string msg
```

- ❖ ret: 执行 SDK 得到的返回值
- ❖ msg: 执行 SDK 得到的返回值的说明

5.3.3.5 Pallet.msg

pallet 输出托盘算法结果相关的信息，pallet 消息格式如下：

```
Header header
int8 status
float32 x
float32 y
float32 yaw
```

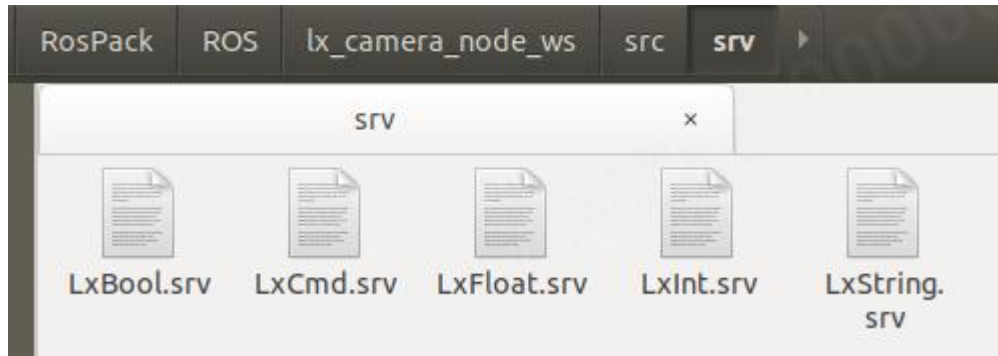
- ❖ header: 当前数据的帧信息以及时间戳
- ❖ status: 托盘识别返回的状态信息:
 - 0: 检测成功
 - 1: 未检测到地面
 - 2: 未检测到托盘腿
 - 3: 托盘距离过远
 - 4: 未知错误
- ❖ x: 托盘距离相机光心的 X 距离
- ❖ y: 托盘距离相机光心的 Y 距离
- ❖ yaw: 托盘中心相对于相机光心的 theta 夹角

节点开始运行后（需开启托盘算法），可以订阅 LxCamera_Pallet 节点获取相关信息，输出如下：

```
---
header:
  seq: 55
  stamp:
    secs: 1711634780
    nsecs: 342781886
  frame_id: "mrdvs"
status: -6
x: 0.0
y: 0.0
yaw: 0.0
---
header:
  seq: 56
  stamp:
    secs: 1711634780
    nsecs: 539528711
  frame_id: "mrdvs"
status: -6
x: -2690.86181641
y: -481.676605225
yaw: 9.25626087189
---
```

5.3.4 特有服务说明

因 ROS1 和 ROS2 特有服务类型一致，所以此处 ROS1 为例。
在 src/srv 文件夹中有如下文件：



这些消息皆是用于设置或者获取相机的某些参数值。

【---】之上是发送的数据，【---】之下是返回的数据

5.3.4.1 LxBool.srv

```
int32 cmd
bool val
bool is_set
---
bool val
Result result
```

- ❖ cmd: 需要获取或者设置的参数编号
- ❖ val: 设置时设置的值
- ❖ is_set: 是否设置参数，为 true 时会把 val 中的值设置到相机中，为 false 时会从相机中把这个参数的值读取并显示出来
- ❖ val: 获取或者设置时返回的相机当前参数的值
- ❖ result: 调用 SDK 接口返回的值和信息

5.3.4.2 LxCmd.srv

```
int32 cmd
---
Result[] result
```

- ❖ cmd: 需要获取或者设置的参数编号，当为 0 时会返回所有的参数编号
- ❖ result: 调用 SDK 接口返回的值和信息，cmd 可能会有多个返回值

5.3.4.3 LxFloat.srv

```
int32 cmd
float32 val
bool is_set
---
float32 max_value
float32 min_value
float32 cur_value
bool available
Result result
```

- ❖ cmd: 需要获取或者设置的参数编号
- ❖ val: 设置时设置的值
- ❖ is_set: 是否设置参数, 为 true 时会将 val 中的值设置到相机中, 为 false 时会从相机中把这个参数的值读取并显示出来
- ❖ max_value: 该参数可设置的最大值
- ❖ min_value: 该参数可设置的最小值
- ❖ cur_value: 该参数在相机中的当前值
- ❖ available: 该参数是否支持设置
- ❖ result: 调用 SDK 接口返回的值和信息

5.3.4.4 LxInt.srv

```
int32 cmd
int32 val
bool is_set
---
int32 max_value
int32 min_value
int32 cur_value
bool available
Result result
```

- ❖ cmd: 需要获取或者设置的参数编号
- ❖ val: 设置时设置的值
- ❖ is_set: 是否设置参数, 为 true 时会将 val 中的值设置到相机中, 为 false 时会从相机中把这个参数的值读取并显示出来
- ❖ max_value: 该参数可设置的最大值
- ❖ min_value: 该参数可设置的最小值
- ❖ cur_value: 该参数在相机中的当前值
- ❖ available: 该参数是否支持设置
- ❖ result: 调用 SDK 接口返回的值和信息

5.3.4.5 LxString.srv

```
int32 cmd
string val
bool is_set
---
string val
Result result
```

- ❖ cmd: 需要获取或者设置的参数编号
- ❖ val: 设置时设置的值
- ❖ is_set: 是否设置参数, 为 true 时会将 val 中的值设置到相机中, 为 false 时会从相机中把这个参数的值读取并显示出来
- ❖ val: 获取或者设置时返回的相机当前参数的值
- ❖ result: 调用 SDK 接口返回的值和信息

5.3.5 主题订阅方式说明

5.3.5.1 订阅帧率, 托盘算法、避障算法输出数据

订阅帧率

在 lx_camera_node_ws 目录下, 输入: bash rate.sh

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ bash rate.sh
```

lx_camera_node 节点图像正常运行并刷新后可以看到如下信息:

```
---
header:
  seq: 101
  stamp:
    secs: 1711634251
    nsecs: 493869994
  frame_id: "mrdvs"
depth: 22.0
amp: 21.0
rgb: 21.0
temperature: 39.6875
```

订阅托盘算法输出

在 lx_camera_node_ws 目录下, 输入: bash pallet.sh

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ bash pallet.sh
```

lx_camera_node 节点图像正常运行并刷新后可以看到如下信息:

```

---
header:
  seq: 55
  stamp:
    secs: 1711634780
    nsecs: 342781886
  frame_id: "mrdvs"
status: -6
x: 0.0
y: 0.0
yaw: 0.0
---
header:
  seq: 56
  stamp:
    secs: 1711634780
    nsecs: 539528711
  frame_id: "mrdvs"
status: -6
x: -2690.86181641
y: -481.676605225
yaw: 9.25626087189
---

```

订阅避障算法输出

在 lx_camera_node_ws 目录下，输入：bash obstacle.sh

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ bash obstacle.sh
```

lx_camera_node 节点图像正常运行并刷新后可以看到如下信息：

```

seq: 25
stamp:
  secs: 1711635011
  nsecs: 949000729
frame_id: "mrdvs"
status: 0
io_output: 2
box_number: 0
box: []
---
header:
  seq: 26
  stamp:
    secs: 1711635012
    nsecs: 137756545
  frame_id: "mrdvs"
status: 0
io_output: 2
box_number: 0
box: []
---

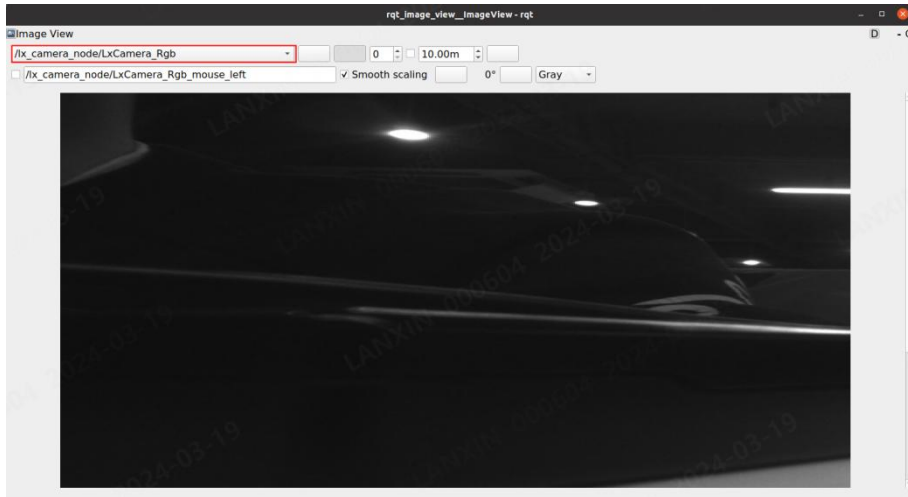
```

5.3.5.2 订阅图像数据（ROS1）

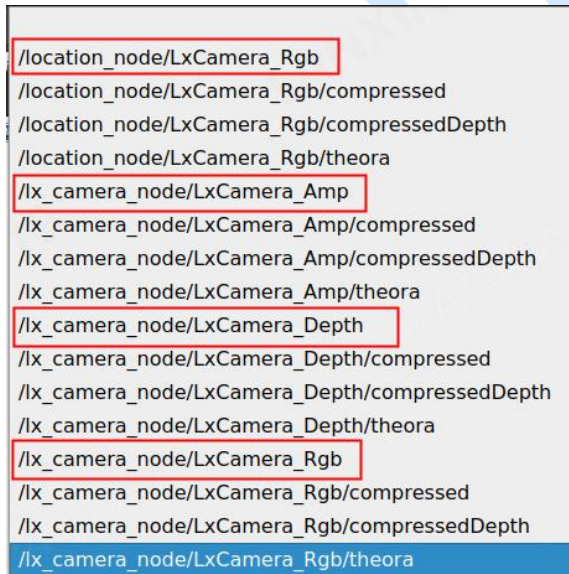
通过 ros 自带的图像工具可以查看节点输出的图像信息：

控制台输入命令：`rqt_image_view`

输入命令后会弹出如下窗口：

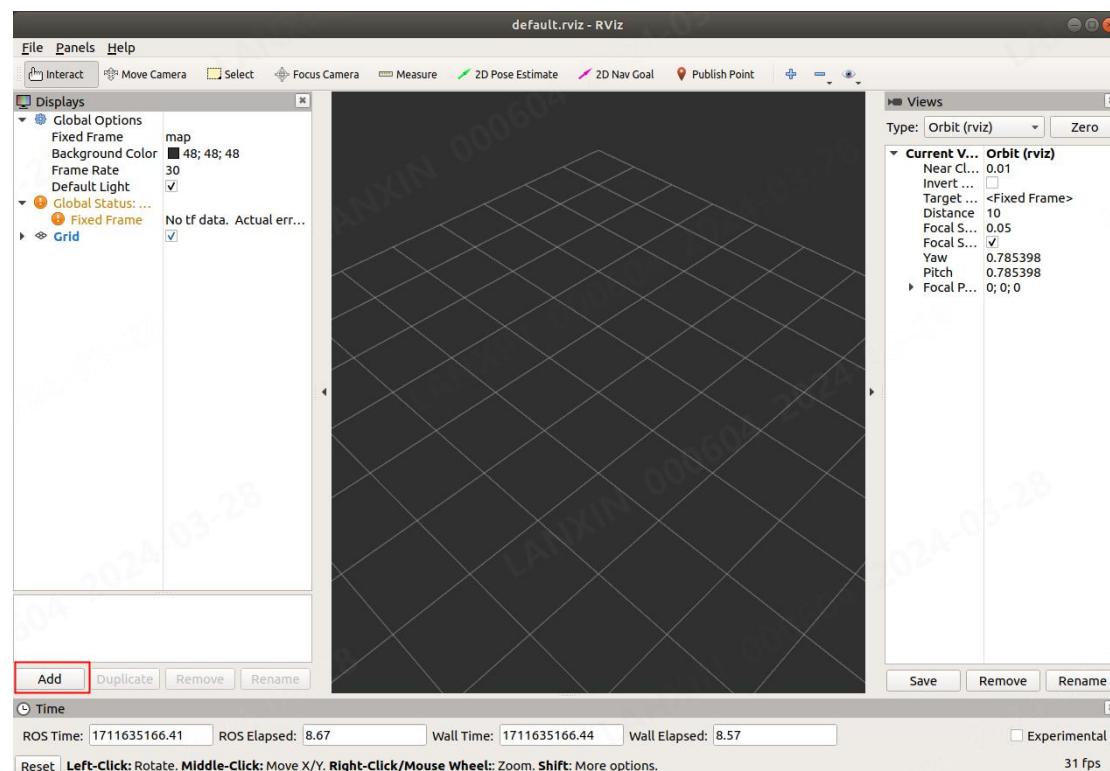


红色框处可以选择，所有节点中的图像，图中展示的是蓝芯相机 rgb 图像，还可以选择深度图或者强度图（前提是流已经开启）：

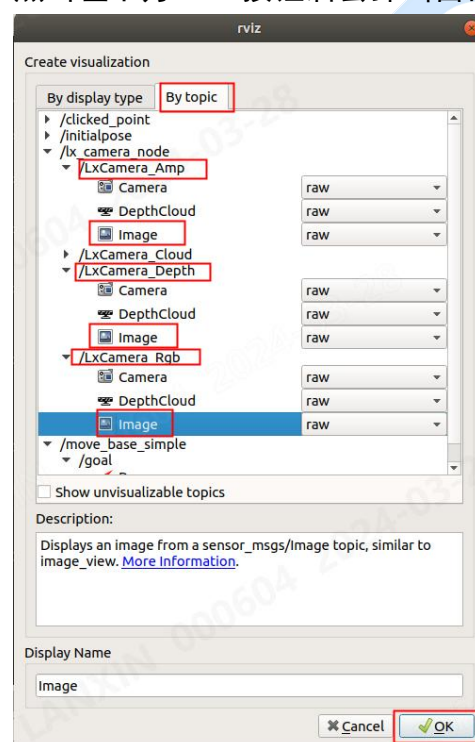


图中，第一个红框是定位算法节点输出的 RGB 信息，其余三个是普通节点输出的图像信息，amp 为强度图，depth 为深度图，rgb 为 rgb 图
选择不同的项可以查看不同的图像

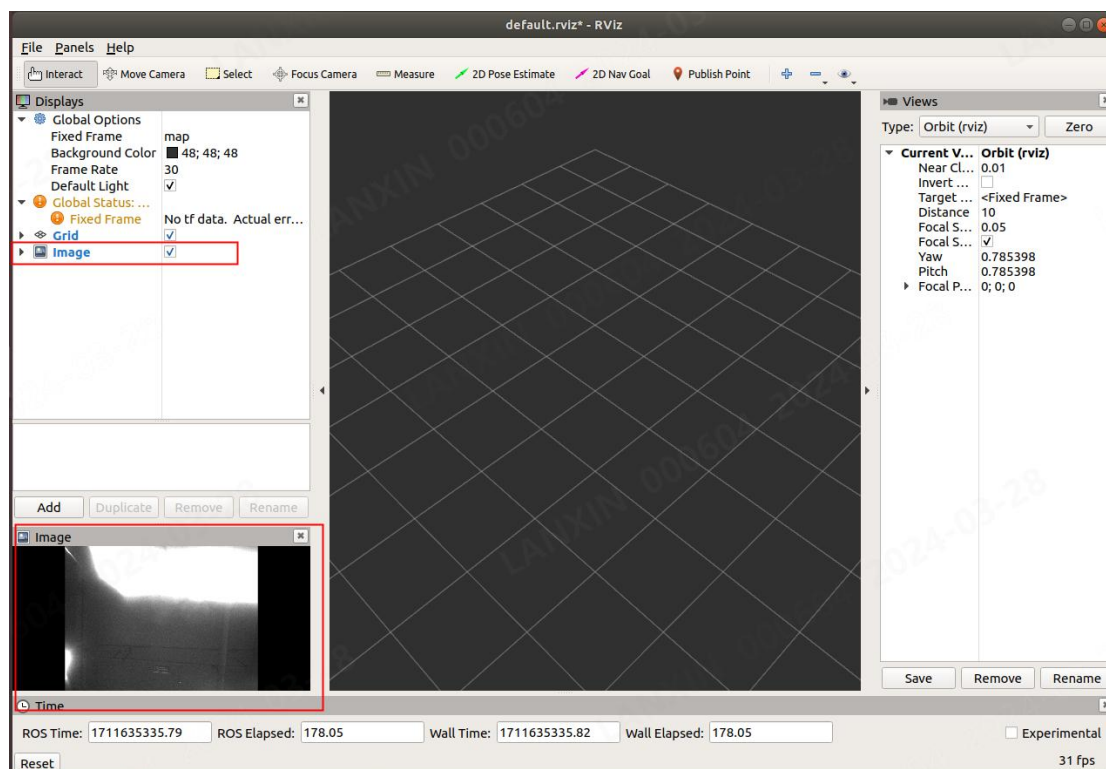
或者使用 `rviz` 查看图像数据，控制台输入命令：`rviz`
会弹出如下窗口：



点击左下方 Add 按钮后会弹出窗口：



选择需要的图像话题并选择其中的 Image 然后点击右下方的 OK，就可以在主界面左侧看到图像数据，如下图显示：

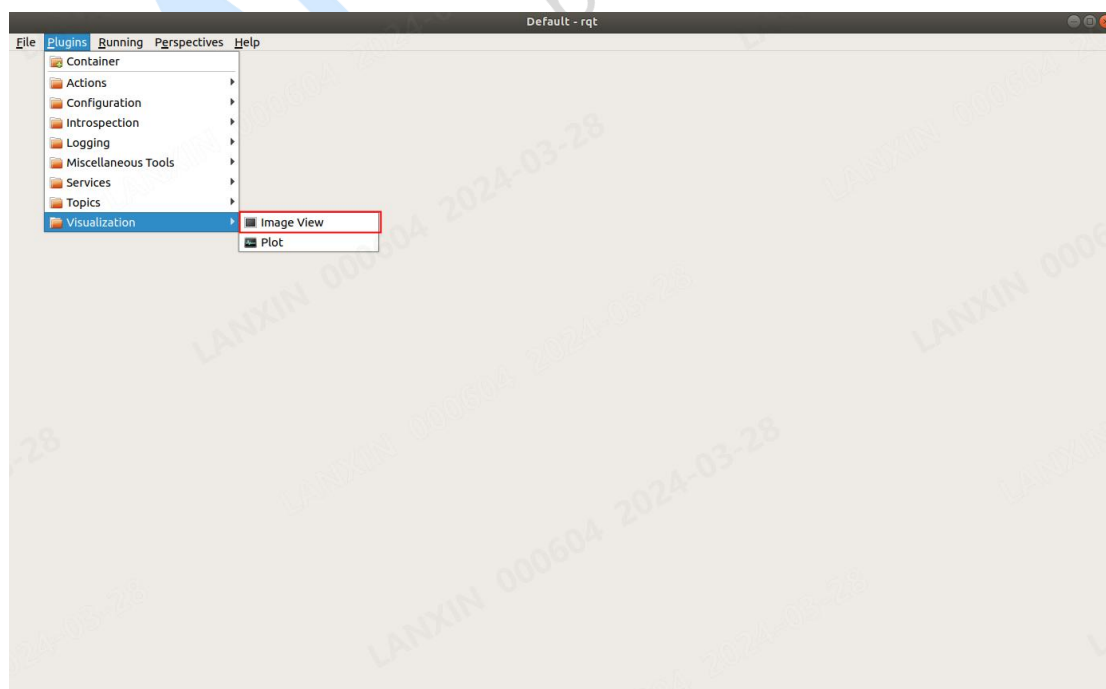


5.3.5.3 订阅图像数据（ROS2）

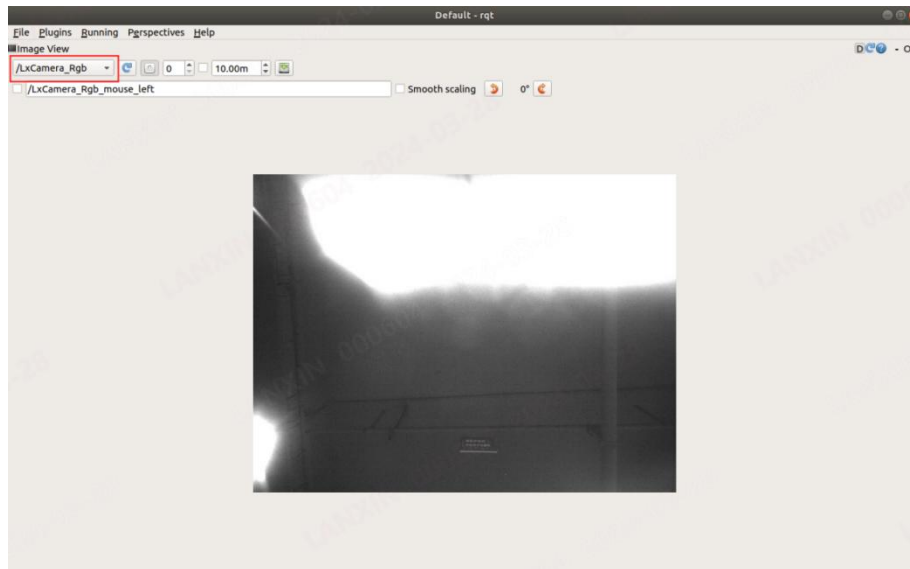
通过 ros2 自带的图像工具可以查看节点输出的图像信息：

控制台输入命令：`rqt`

输入命令后会弹出如下窗口：



选择菜单栏第二项最底下选项的第一个 Image View，选择后会弹出如下界面：

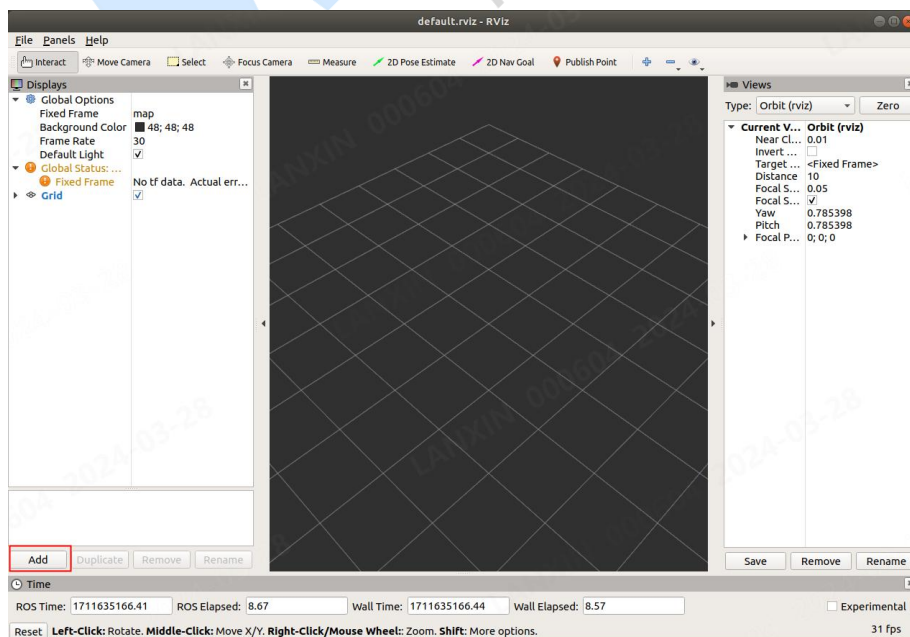


红色框处可以选择，所有节点中的图像，图中展示的是蓝芯相机 rgb 图像，还可以选择深度图或者强度图（前提是流已经开启）：

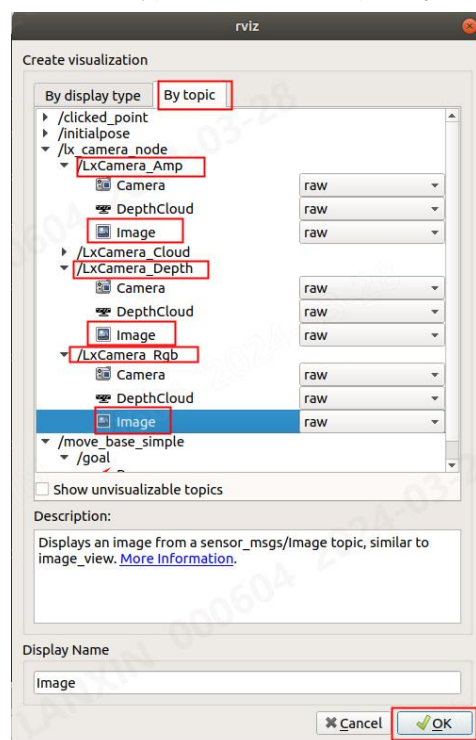


图中，第一个红框是定位算法节点输出的 RGB 信息，其余三个是普通节点输出的图像信息，amp 为强度图，depth 为深度图，rgb 为 rgb 图
选择不同的项可以查看不同的图像

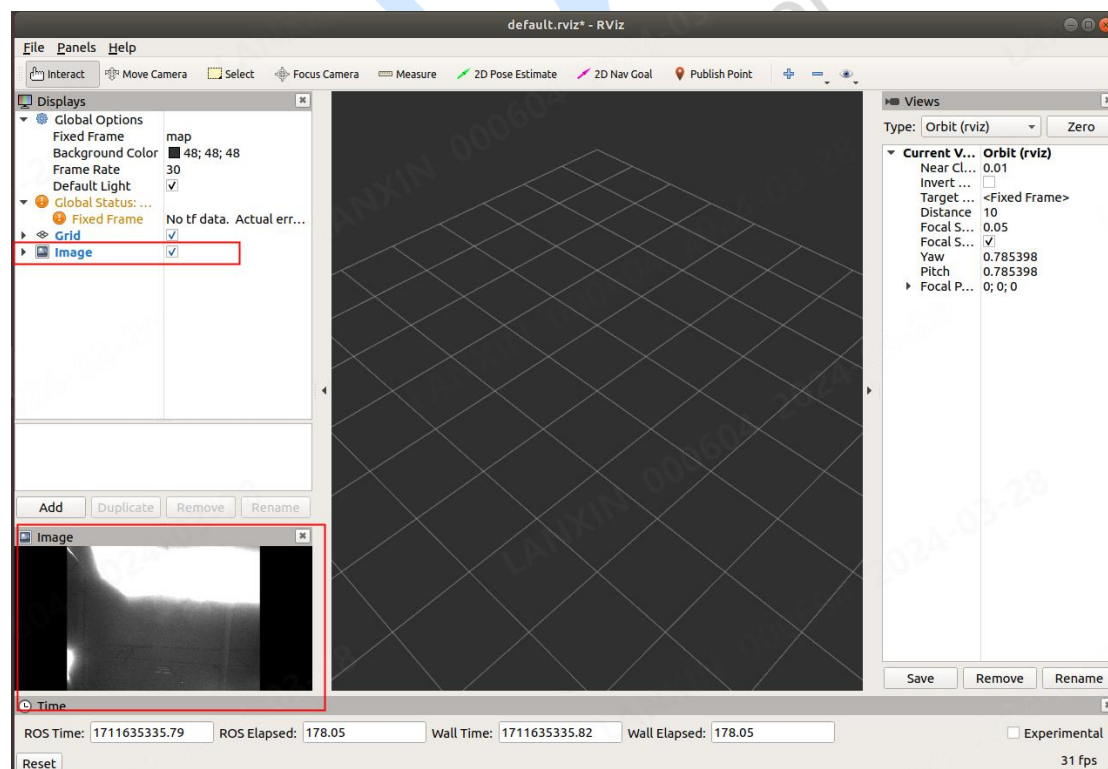
或者使用 rviz2 查看图像数据，控制台输入命令：`rviz2`
会弹出如下窗口：



点击左下方 Add 按钮后会弹出窗口：



选择需要的图像话题并选择其中的 Image 然后点击右下方的 OK，就可以在主界面左侧看到图像数据，如下图所示：



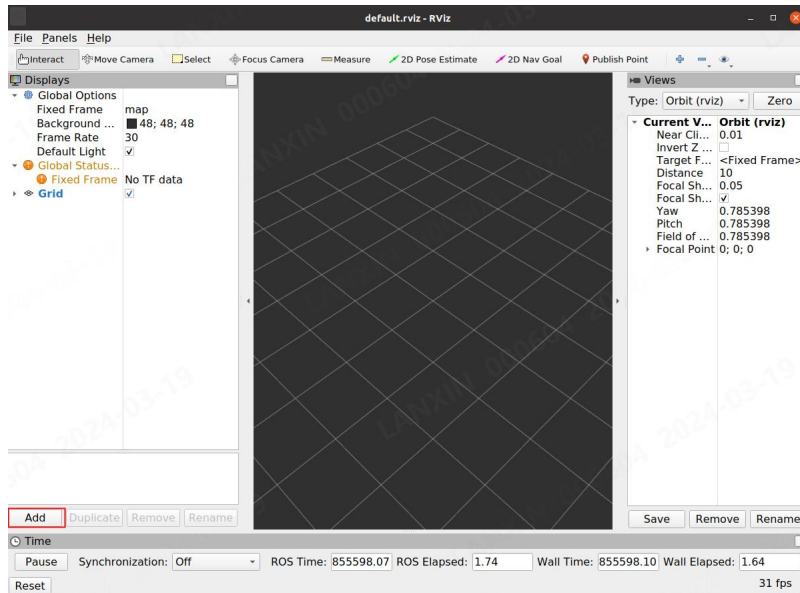
部分系统需要在上图【map】处填入【mrdvs】才能看到图像

5.3.5.4 订阅点云数据

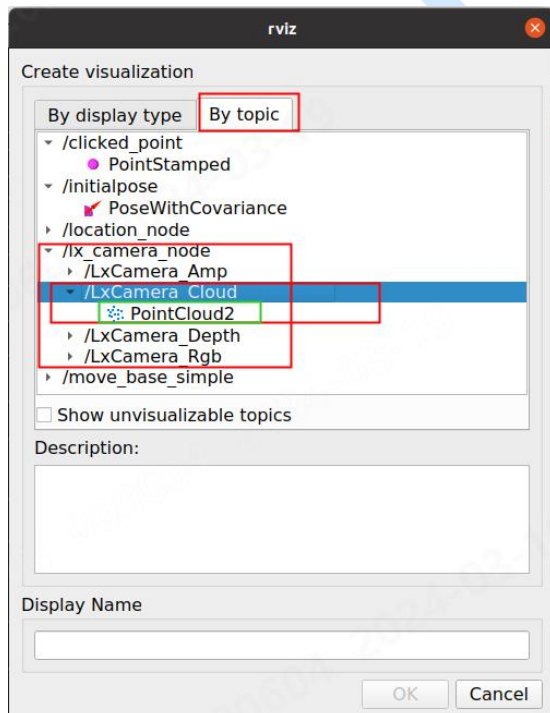
通过 ros 或者 ros2 自带的点云工具可以查看节点输出的点云信息：

命令：rviz【ROS1】，rviz2【ROS2】

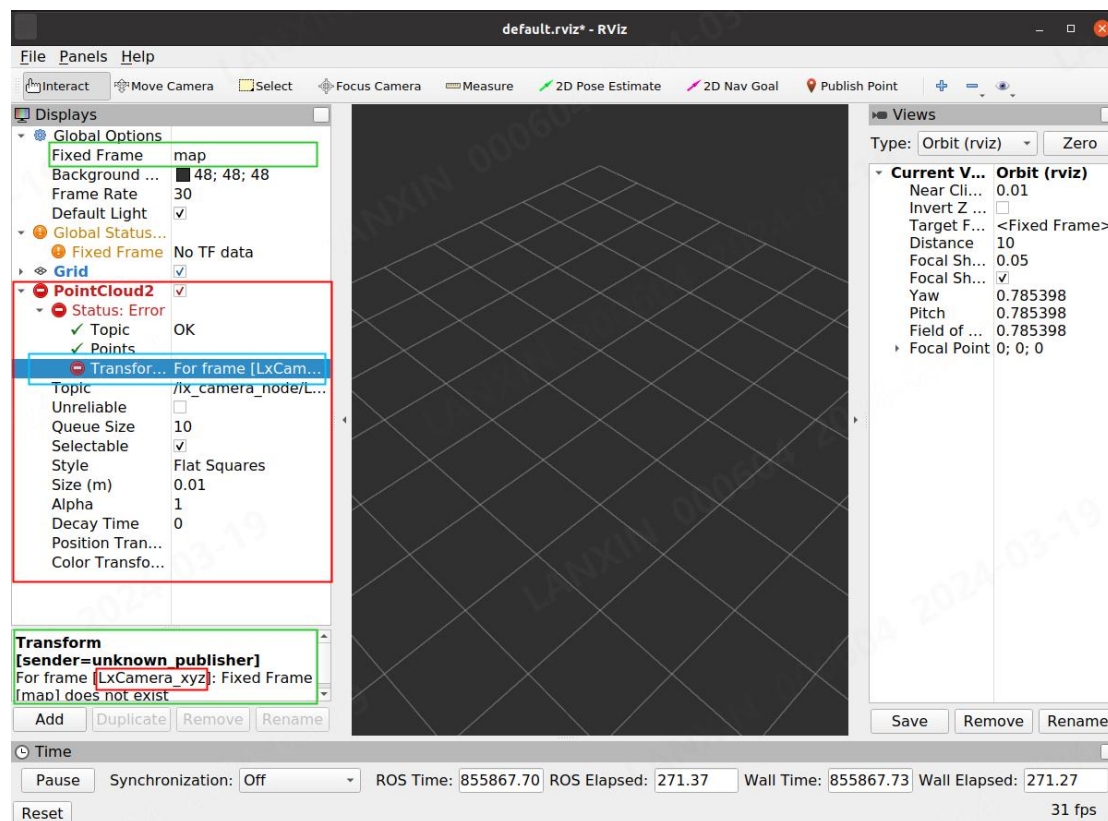
输入命令后会弹出如下窗口：



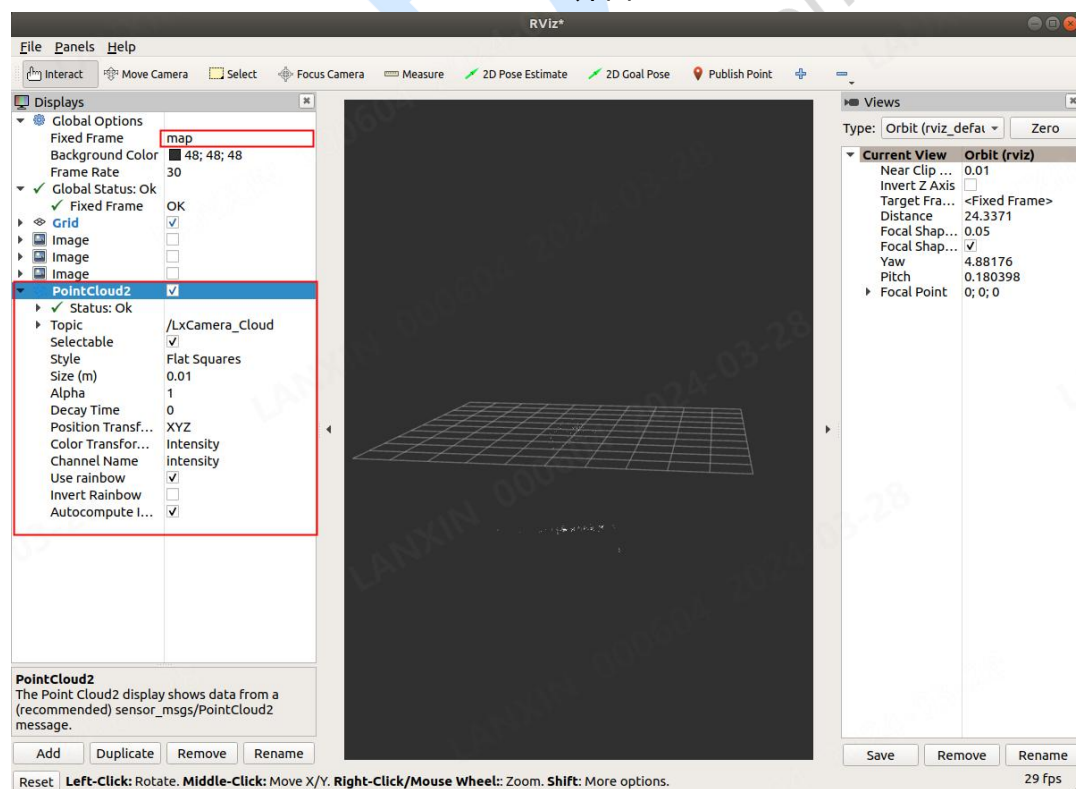
点击下方红色框内按钮会弹出如下框：



选择第二个页签，会有名为 lx_camera_node 的节点，节点中有名为 LxCamera_Cloud 的主题，主题中有名为 PointCloud2 的消息，选择后点击 OK 即可，完成后界面如下：

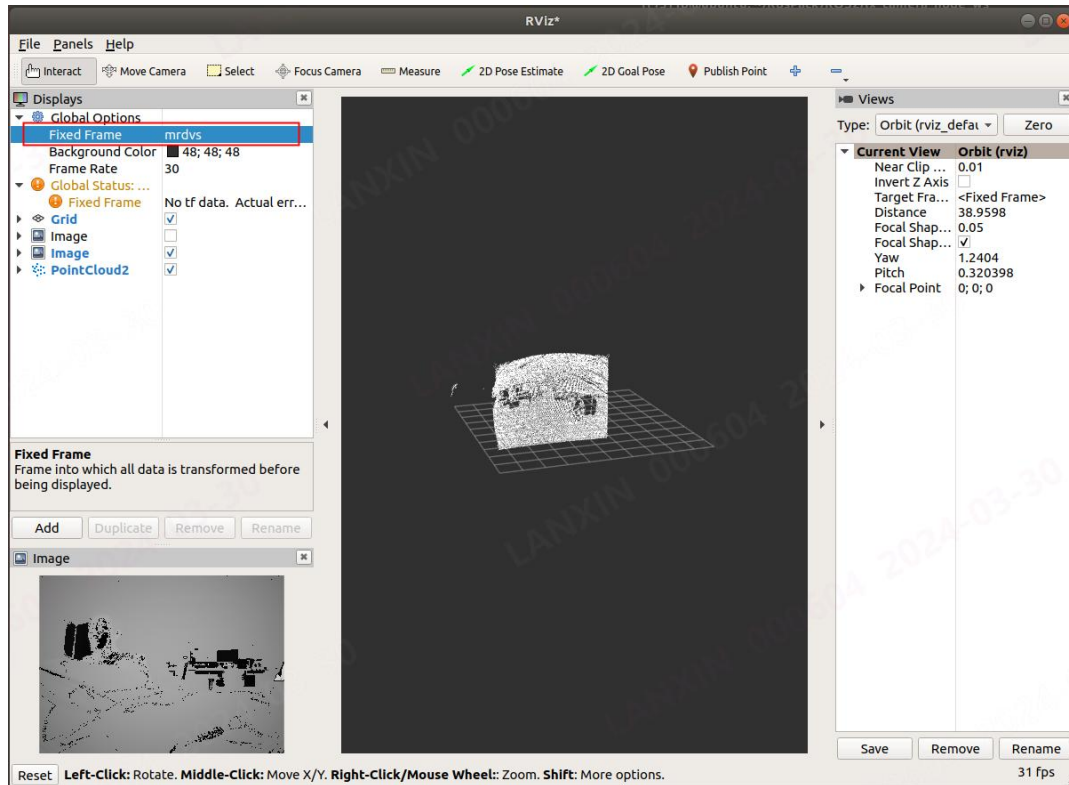


(ROS1 界面)



(ROS2 界面)

此时点云还不能正常显示，在上方红绿框 `map` 处填入 `mrdvs` 就可以看到点云，效果如下：



ROS 中默认的点云单位为米，所以使用 rviz 查看时需要将 launch 文件中的 `lx_tof_unit` 修改为 1【默认即为 1】

5.3.6 服务通讯方式说明

在 ROS1 中所有的命令都可以用 Tab 键补全，但是在 ROS2 中只有主题能被补全，消息部分需要手动输入。

ROS1 输入数据时按照补全格式输入即可，ROS2 中输入数据时需要按照 yaml 文件的格式来输入。

节点运行之后，在 `lx_camera_node_ws` 文件夹下打开一个终端，在终端中配置环境，输入命令：`source devel/setup.bash`

5.3.6.1 获取所有参数编号列表 (Lx_Cmd)

ros1

在终端输入：`rosservice call /lx_camera_node/LxCamera_LxCmd "cmd: 0"`

```
lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxCmd "cmd: 0"
```

可以看到输出：

```

result:
-
  ret: 1001
  msg: "INT" FIRST_EXPOSURE ""
-
  ret: 1002
  msg: "INT" SECOND_EXPOSURE ""
-
  ret: 1003
  msg: "INT" THIRD_EXPOSURE ""
-
  ret: 1004
  msg: "INT" FOURTH_EXPOSURE ""
-
  ret: 1005
  msg: "INT" GAIN ""
-
  ret: 1011
  msg: "INT" MIN_DEPTH ""
-
  ret: 1012
  msg: "INT" MAX_DEPTH ""
-
  ret: 1013
  msg: "INT" MIN_AMPLITUDE ""
-
  ret: 1014
  msg: "INT" MAX_AMPLITUDE ""
-
  ret: 1016
  msg: "INT" CODE_MODE ""
-
  ret: 1018

```

1012	INT	MAX_DEPTH	最大深度值
1013	INT	MIN_AMPLITUDE	有效信号最小强度值
1014	INT	MAX_AMPLITUDE	有效信号最大强度值
1016	INT	CODE_MODE	编码模式
1018	INT	WORK_MODE	工作模式
1019	INT	LINK_SPEED	网卡网速
1021	INT	3D_IMAGE_WIDTH	3D 图像分辨率宽度
1022	INT	3D_IMAGE_HEIGHT	3D 图像分辨率高度
1023	INT	3D_IMAGE_OFFSET_X	ROI 水平偏移像素
1024	INT	3D_IMAGE_OFFSET_Y	ROI 垂直偏移像素
1025	INT	3D_BINNING_MODE	3D 图像 binning
1026	INT	3D_DEPTH_DATA_TYPE	深度图像数据格式
1031	INT	3D_AMPLITUDE_CHANNEL	3D 强度图像通道数
1032	INT	3D_AMPLITUDE_GET_TYPE	获取强度图像的方式
1033	INT	3D_AMPLITUDE_EXPOSURE	3D 手动曝光值
1034	INT	3D_AMPLITUDE_INTENSITY	3D 自动目标亮度
1035	INT	3D_AMPLITUDE_DATA_TYPE	强度图像数据格式
1036	INT	3D_AUTO_EXPOSURE_LEVEL	3D 自动曝光等级
1041	INT	2D_IMAGE_WIDTH	2D 图像分辨率宽度
1042	INT	2D_IMAGE_HEIGHT	2D 图像分辨率高度
1043	INT	2D_IMAGE_OFFSET_X	2D 图像 ROI 水平偏移像素
1044	INT	2D_IMAGE_OFFSET_Y	2D 图像 ROI 垂直偏移像素
1045	INT	2D_BINNING_MODE	2D 图像 binning
1046	INT	2D_IMAGE_CHANNEL	2D 图像通道数
1047	INT	2D_IMAGE_DATA_TYPE	2D 图像数据格式
1051	INT	2D_MANUAL_EXPOSURE	2D 手动曝光时的曝光值
1052	INT	2D_MANUAL_GAIN	2D 手动曝光时的增益值
1053	INT	2D_ENCODE_TYPE	2D 图像压缩格式
1054	INT	2D_AUTO_EXPOSURE_LEVEL	2D 图像自动曝光等级
1061	INT	TOF_GLOBAL_OFFSET	TOF 深度数据偏移
1062	INT	3D_UNDISTORT_SCALE	TOF 图像反畸变系数
1065	INT	ALGORITHM_MODE	设置内置应用算法
1066	INT	MODBUS_ADDR	modbus 地址
1067	INT	HEART_TIME	与设备间心跳时间
1068	INT	GVSP_PACKET_SIZE	GVSP 单包数据分包大小
1069	INT	TRIGGER_MODE	触发模式
1070	INT	CALCULATE_UP	算法上下移
1072	INT	CAN_BAUD_RATE	can 的波特率值
1073	INT	MAX_3D_AUTO_EXPOSURE	3d 自动曝光上限
1074	INT	CUSTOM_PARAM_GROUP	用户自定义参数组
2001	FLOAT	FILTER_LEVEL	滤波等级
2002	FLOAT	EST_OUT_EXPOSURE	是否评估过曝数据
2003	FLOAT	LIGHT_INTENSITY	光强度
2004	FLOAT	3D_DEPTH_FPS	深度图当前帧率

2005	FLOAT	3D_AMPLITUDE_FPS	强度图当前帧率
2006	FLOAT	2D_IMAGE_FPS	RGB 图当前帧率
2007	FLOAT	DEVICE_TEMPERATURE	相机当前温度
3001	BOOL	CONNECT_STATE	当前连接状态
3002	BOOL	ENABLE_3D_DEPTH_STREAM	开启/关闭深度数据流
3003	BOOL	ENABLE_3D_AMP_STREAM	开启/关闭强度数据流
3006	BOOL	ENABLE_3D_AUTO_EXPOSURE	3D 自动曝光使能
3007	BOOL	ENABLE_3D_UNDISTORT	3D 反畸变使能
3008	BOOL	ENABLE_ANTI_FLICKER	抗频闪使能
3011	BOOL	ENABLE_2D_STREAM	开启/关闭 2D 数据流
3012	BOOL	ENABLE_2D_AUTO_EXPOSURE	2D 自动曝光使能
3015	BOOL	ENABLE_2D_UNDISTORT	2D 图像反畸变使能
3016	BOOL	ENABLE_2D_TO_DEPTH	2D3D 图像对齐使能
3017	BOOL	ENABLE_BACKGROUND_AMP	强度背景光使能
3018	BOOL	ENABLE_MULTI_MACHINE	多机模式使能
3019	BOOL	ENABLE_MULTI_EXPOSURE_HDR	HDR
3020	BOOL	ENABLE_SYNC_FRAME	是否开启强制帧同步
4001	STRING	DEVICE_VERSION	设备版本号
4002	STRING	DEVICE_LOG_NAME	日志文件名
4003	STRING	FIRMWARE_NAME	固件文件名
4004	STRING	FILTER_PARAMS	滤波算法参数
4005	STRING	ALGORITHM_PARAMS	内置算法参数
4006	STRING	ALGORITHM_VERSION	内置算法版本号
4007	STRING	DEVICE_OS_VERSION	设备系统镜像版本号
0	CMD	GET_PARAM_LIST	获取所有参数编号
1	CMD	START_STREAM	设备启流
2	CMD	STOP_STREAM	设备停流
5001	CMD	GET_NEW_FRAME	更新当前最新数据
5002	CMD	RETURN_VERSION	回退上一版本
5003	CMD	RESTART_DEVICE	重启相机
5004	CMD	WHITE_BALANCE	自动白平衡
5007	CMD	RESET_PARAM	恢复默认参数

部分参数需要先停流才能设置，或者设置有先后顺序，具体需求请跟技术支持人员对接。

5.3.6.2 获取 Int 参数 (Lx_Int)

ROS1

```
在终端输入: rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001
val: 0
is_set: false"
```

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001
val: 0
is_set: false"
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001
val: 0
is_set: false"
max_value: 3600
min_value: 1
cur_value: 1010
available: True
result:
  ret: 0
  msg: " Success"
```

红框内的就是返回值，图中获取的是高积分时间，根据返回结果可以看到，最大值是 **3600**，最小值是 **1**，当前值是 **1010**，是否支持设置是**支持**，接口调用返回值是 **0**，返回值信息是**成功**。

ROS2

在终端输入：`ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 0,is_set: false}'`

```
ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 0,is_set: false}'
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 0,is_set: false}'
requester: making request: lx_camera_ros.srv.LxInt_Request(cmd=1001, val=0, is_set=False)
response:
lx_camera_ros.srv.LxInt_Response(max_value=3600, min_value=1, cur_value=1010, available=True, result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中获取的是高积分时间，根据返回结果可以看到，最大值是 **3600**，最小值是 **1**，当前值是 **1010**，是否支持设置是**支持**，接口调用返回值是 **0**，返回值信息是**成功**。

5.3.6.3 设置 Int 参数 (Lx_Int)

ROS1

在终端输入：`rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001`

`val: 2000`

`is_set: true"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001
val: 2000
is_set: true"
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxInt "cmd: 1001
val: 2000
is_set: true"
max_value: 3600
min_value: 1
cur_value: 2000
available: True
result:
  ret: 0
  msg: " Success"
```

红框内的就是返回值，图中设置的是高积分时间，根据返回结果可以看到，最大值是 **3600**，最小值是 **1**，当前值已经修改为 **2000**，是否支持设置是**支持**，接口调用返回值是 **0**，返回值信息是**成功**。

ROS2

在终端输入：`ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 3000,is_set: true}'`

```
ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 3000,is_set: true}'
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxInt lx_camera_ros/srv/LxInt '{cmd: 1001,val: 3000,is_set: true}'
waiting for service to become available...
requester: making request: lx_camera_ros.srv.LxInt_Request(cmd=1001, val=3000, is_set=True)

response:
lx_camera_ros.srv.LxInt_Response(max_value=3600, min_value=1, cur_value=3000, available=True, result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中获取的是高积分时间，根据返回结果可以看到，最大值是 **3600**，最小值是 **1**，当前值已经修改为 **3000**，是否支持设置是**支持**，接口调用返回值是 **0**，返回值信息是**成功**。

5.3.6.4 获取 Float 参数 (Lx_Float)

ROS1

在终端输入: `rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007`

`val: 0.0`

`is_set: false"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007"
val: 0.0
is_set: false"
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007"
val: 0.0
is_set: false"
max_value: 120.0
min_value: -40.0
cur_value: 49.6875
available: False
result:
  ret: 0
  msg: " Success"
```

红框内的就是返回值, 图中获取的是相机温度, 根据返回结果可以看到, 最大值是 **120**, 最小值是 **-40**, 当前值是 **49.6875**, 是否支持设置是**不支持**, 接口调用返回值是 **0**, 返回值信息是**成功**。

ROS2

在终端输入: `ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 0,is_set: false}'`

```
ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 0,is_set: false}'
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 0,is_set: false}'
requester: making request: lx_camera_ros.srv.LxFloat_Request(cmd=2007, val=0.0, is_set=False)
response:
lx_camera_ros.srv.LxFloat_Response(max_value=120.0, min_value=-40.0, cur_value=50.4375, available=False, result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中获取的是相机温度, 根据返回结果可以看到, 最大值是 **120**, 最小值是 **-40**, 当前值是 **50.4375**, 是否支持设置是**不支持**, 接口调用返回值是 **0**, 返回值信息是**成功**。

5.3.6.5 设置 Float 参数 (Lx_Float)

ROS1

在终端输入: `rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007`

`val: 5.0`

`is_set: true"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007
val: 5.0
is_set: true"
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxFloat "cmd: 2007
val: 5.0
is set: true"
max_value: 120.0
min_value: -40.0
cur_value: 50.9375
available: False
result:
  ret: -2
  msg: " Not Support"
```

红框内的就是返回值，图中设置的是相机温度，根据返回结果可以看到，最大值是 120，最小值是-40，当前值是 50.9375，是否支持设置是不支持，接口调用返回值是-2，返回值信息是不支持，此处设置的值是 5，当前值和设置值不一致是因为该参数不支持设置。

ROS2

在终端输入: `ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 5.0,is_set: true}'`

```
ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 5.0,is_set: true}'
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxFloat lx_camera_ros/srv/LxFloat '{cmd: 2007,val: 5.0,is_set: true}'
requester: making request: lx_camera_ros.srv.LxFloat_Request(cmd=2007, val=5.0, is_set=True)
{response:
  lx_camera_ros.srv.LxFloat_Response(max_value=120.0, min_value=-40.0, cur_value=51.125, available=False, result=lx_camera_ros.msg.Result(ret=-2, msg=' Not Support'))}
```

图中设置的是相机温度，根据返回结果可以看到，最大值是 120，最小值是-40，当前值是 51.125，是否支持设置是不支持，接口调用返回值是-2，返回值信息是不支持，此处设置的值是 5，当前值和设置值不一致是因为该参数不支持设置。

5.3.6.6 获取 Bool 参数 (Lx_Bool)

ROS1

在终端输入：`rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012`

`val: false`

`is_set: false"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012"
val: false
is_set: false"
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012"
val: false
is_set: false"
val: False
result:
  ret: 0
  msg: " Success"
```

红框内的就是返回值，图中获取的是 2D 自动曝光，根据返回结果可以看到，当前值是 **false**，接口调用返回值是 **0**，返回值信息是**成功**。

ROS2

在终端输入：`ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: false,is_set: false}'`

```
ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: false,is_set: false}'
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: false,is_set: false}'
waiting for service to become available...
requester: making request: lx_camera_ros.srv.LxBool_Request(cmd=3012, val=False, is_set=False)

response:
lx_camera_ros.srv.LxBool_Response(val=False, result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中获取的是 2D 自动曝光，根据返回结果可以看到，当前值是 **false**，接口调用返回值是 **0**，返回值信息是**成功**。

5.3.6.7 设置 Bool 参数 (Lx_Bool)

ROS1

在终端输入：`rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012`

`val: true`

`is_set: true"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012
val: true
is_set: true"
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxBool "cmd: 3012
val: true
is_set: true"
val: True
result:
  ret: 0
  msg: " Success"
```

红框内的就是返回值，图中设置的是 2D 自动曝光，根据返回结果可以看到，当前值是 **true**，接口调用返回值是 **0**，返回值信息是**成功**，当前值和设置值一致。

ROS2

在终端输入：`ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: true,is_set: true}'`

```
ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: true,is_set: true}'
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxBool lx_camera_ros/srv/LxBool '{cmd: 3012,val: true,is_set: true}'
waiting for service to become available...
requester: making request: lx_camera_ros.srv.LxBool_Request(cmd=3012, val=True, is_set=True)

response:
lx_camera_ros.srv.LxBool_Response(val=True, result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中设置的是 2D 自动曝光，根据返回结果可以看到，当前值是 **true**，接口调用返回值是 **0**，返回值信息是**成功**，当前值和设置值一致。

5.3.6.8 获取 String 参数 (Lx_String)

ROS1

在终端输入: `rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001
val: ''
is_set: false"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001  
val: ''  
is_set: false"
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001  
val: ''  
is_set: false"  
val: "V1.1.33_231201"  
result:  
  ret: 0  
  msg: " Success"
```

红框内的就是返回值, 图中获取的是设备版本号, 根据返回结果可以看到, 当前值是 **V1.1.33_231201**, 接口调用返回值是 **0**, 返回值信息是**成功**。

ROS2

在终端输入: `ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd: 4001,val: '',is_set: false}'`

```
ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd: 4001,val: '',is_set: false}'
```

可以看到输出:

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd: 4001,val: '',is_set: false}'  
requester: making request: lx_camera_ros.srv.LxString_Request(cmd=4001, val='None', is_set=False)  
  
response:  
lx_camera_ros.srv.LxString_Response(val='V1.1.33_231201', result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中获取的是设备版本号, 根据返回结果可以看到, 当前值是 **V1.1.33_231201**, 接口调用返回值是 **0**, 返回值信息是**成功**。

5.3.6.9 设置 String 参数 (Lx_String)

ROS1

在终端输入：`rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001
val: 'put:\'123\''
is_set: true"`

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001  
val: 'put:\'123\''  
is_set: true"
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS/lx_camera_node_ws$ rosservice call /lx_camera_node/LxCamera_LxString "cmd: 4001  
val: 'put:\'123\''  
is_set: true"  
val: ''  
result:  
ret: -2  
msg: " Not Support"
```

红框内的就是返回值，图中设置的是设备版本号，根据返回结果可以看到，当前值是 **V1.1.33_231201**，接口调用返回值是-2，返回值信息是不支持，当前值和设置值不一致，因为该参数不支持设置。

ROS2

在终端输入：`ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd:
4001,val: "put:\'123\''',is_set: true}'`

```
ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd: 4001,val: "put:\'123\''',is_set: true}'
```

可以看到输出：

```
fr1511b@ubuntu:~/RosPack/ROS2/lx_camera_node_ws$ ros2 service call /LxCamera_LxString lx_camera_ros/srv/LxString '{cmd: 4001,val: "put:\'123\''',is_set: true}'  
requester: making request: lx_camera_ros.srv.LxString_Request(cmd=4001, val='put:\'123\''', is_set=True)  
response:  
lx_camera_ros.srv.LxString_Response(val='V1.1.33_231201', result=lx_camera_ros.msg.Result(ret=0, msg=' Success'))
```

图中设置的是设备版本号，根据返回结果可以看到，当前值是 **V1.1.33_231201**，接口调用返回值是-2，返回值信息是不支持，当前值和设置值不一致，因为该参数不支持设置。。

在设置 String 类型的值时，对于单引号和双引号一定要加转义符，除了 ROS2 中一开始的双引号和 ROS1 中一开始的单引号，例如要输入字符串【PUT"123"】就必须加入转义符变成：【PUT\'123\'】。

六、ROS 推荐硬件配置与资源占用

6.1 推荐硬件配置

在推荐硬件配置中，会根据 CUP 的内核数量、CPU 占用率、内存大小进行分析，此处给出的配置皆为推荐配置，不是最低配置，如果硬件配置低于推荐配置，则可以通过减少流输出、关闭点云计算、算法下移等操作来实现稳定运行。在运行所需配置高于现有配置时，可能导致 ROS 核心崩溃甚至 Ubuntu 系统崩溃。详细的使用情况会在本章第二、第三小节中给出，请根据情况选择最合适的运行环境和流配置。此处所有数据均测试于 VM 虚拟机，测试相机为 M4PRO。

Ubuntu16.04

ROS 版本	CPU	内存	带宽
ros	AMD64 【四核】	4G	1000M

Ubuntu18.04

ROS 版本	CPU	内存	带宽
ros	AMD64 【四核】	4G	1000M
ros2	AMD64 【四核】	4G	1000M

Ubuntu20.04

ROS 版本	CPU	内存	带宽
ros	AMD64 【四核】	4G	1000M
ros2	AMD64 【四核】	4G	1000M

Ubuntu22.04

ROS 版本	CPU	内存	带宽
ros2	AMD64 【四核】	4G	1000M

6.2 CPU 详情

在 CPU 详情中会展示该节点在不同运行情况下所占用的 CPU 资源数量，测试基于 AMD64【8 核】，8G 内存硬件环境下，测试相机为 M4PRO，测试算法为托盘算法，2D 宽高为 1280*960，3D 宽高为 640*480、2D 反畸变开启、3D 反畸变关闭、积分 110 - 3000、深度 0 - 8000、TOF 帧率 14、流模式、平滑等级 1、噪声等级 1、时域等级 0，统计指令为 TOP。因测试软环境是 VM 虚拟机，因此实体机所展示的数据与文档中相比有部分波动属于正常现象。对于部分相机，如 S2，会有部分算法在上位机计算，所以导致资源占用量大，属于正常现象。

部分名词说明如下：

- ❖ 单流：指开启 RGB、深度、强度任意一个流
- ❖ 双流：指开启 RGB、深度、强度任意两个流
- ❖ 三流：指 RGB、深度、强度三个流全部开启
- ❖ 四流：指 RGB、深度、强度、点云全部开启

算法于部分流属于绑定关系，绑定关系如下：

- ❖ 开启托盘算法则必须开启深度
- ❖ 开启避障算法则必须开启深度
- ❖ 开启定位算法则必须开启 RGB

表格中浅蓝色背景为开启点云的运行配置，无色背景为不开启点云的运行配置。以下所有测试数据默认为算法下移，因此对于 S2 系列相机来说，所有测试数值可能偏小【S2 系列相机部分算法强制上移】。

Ubuntu16.04

ROS1		
运行配置	均值%	区间%
单流	10.59	0.0~26.7
双流	14.13	0.0~60.0
三流	16.56	0.0~37.5
四流	108.25	93.8~162.5
点云+深度	105.13	93.8~153.3
算法+深度	5.44	0.0~43.0
算法+深度+点云	104.39	93.8~120.0
算法+深度+RGB	13.15	6.2~40.0
算法+深度+RGB+点云	105.96	93.8~146.7
算法+深度+强度+RGB+点云	109.29	93.8~162.5

Ubuntu18.04

ROS1		
运行配置	均值%	区间%
单流	10.31	0.0~27.5
双流	12.81	5.9~27.8
三流	16.23	5.9~37.5
四流	105.95	88.2~131.2
点云+深度	103.55	88.2~156.2
算法+深度	6.32	0.0~43.8
算法+深度+点云	104.07	78.3~158.8
算法+深度+RGB	15.22	5.9~70.6
算法+深度+RGB+点云	104.56	93.8~125.0
算法+深度+强度+RGB+点云	105.52	61.5~204.8

ROS2		
运行配置	均值%	区间%
单流	19.19	5.9~88.2
双流	23.17	6.2~109.1
三流	30.59	9.1~125.0
四流	108.44	44.4~158.6
点云+深度	102.19	37.5~143.9
算法+深度	6.44	0.0~26.7
算法+深度+点云	104.45	79.2~176.5
算法+深度+RGB	14.82	5.9~50.0
算法+深度+RGB+点云	106.89	94.1~137.5
算法+深度+强度+RGB+点云	108.41	94.1~162.5

Ubuntu20.04

ROS1		
运行配置	均值%	区间%
单流	11.46	0.0~50.0
双流	14.62	6.2~66.7
三流	18.46	6.2~50.0
四流	43.64	25.0~143.8
点云+深度	28.37	12.5~75.0
算法+深度	6.53	0.0~43.8
算法+深度+点云	29.28	13.3~100
算法+深度+RGB	13.79	0.0~50.0
算法+深度+RGB+点云	40.64	25.0~81.2
算法+深度+强度+RGB+点云	44.44	25.0~75.0

ROS2		
运行配置	均值%	区间%
单流	8.07	0.0~20.0
双流	11.25	0.0~33.3
三流	16.18	6.2~62.5
四流	33.77	0.0~86.7
点云+深度	27.45	12.5~81.2
算法+深度	6.13	0.0~68.8
算法+深度+点云	31.11	12.5~56.2
算法+深度+RGB	11.75	0.0~68.8
算法+深度+RGB+点云	38.04	6.2~75.0
算法+深度+强度+RGB+点云	40.85	18.8~106.2

Ubuntu22.04

ROS1		
运行配置	均值%	区间%
单流	8.05	0.0~23.5
双流	11.99	0.0~75.0
三流	18.08	0.0~100
四流	50.06	6.2~200
点云+深度	33.92	0.0~106.2
算法+深度	7.70	0.0~43.8
算法+深度+点云	36.20	6.2~131.2
算法+深度+RGB	14.46	6.2~37.5
算法+深度+RGB+点云	39.07	18.8~94.1
算法+深度+强度+RGB+点云	45.61	12.5~93.8

6.3 内存详情

在内存详情中会展示该节点在不同运行情况下所占用内存资源数量，测试基于 **AMD64【8核】**，**8G 内存** 硬件环境下，测试相机为 **M4PRO**，测试算法为**托盘算法**，**2D 宽高为 1280*960**，**3D 宽高为 640*480**、**2D 反畸变开启**、**3D 反畸变关闭**、**积分 110 - 3000**、**深度 0 - 8000**、**TOF 帧率 14**、**流模式**、**平滑等级 1**、**噪声等级 1**、**时域等级 0**，统计指令为 **TOP**。因测试软环境是 **VM 虚拟机**，因此实体机所展示的数据与文档中相比有部分波动属于正常现象。对于部分相机，如 **S2**，会有部分算法在上位机计算，所以导致资源占用量大，属于正常现象。

部分名词说明如下：

- ❖ 单流：指开启 RGB、深度、强度任意一个流
- ❖ 双流：指开启 RGB、深度、强度任意两个流
- ❖ 三流：指 RGB、深度、强度三个流全部开启
- ❖ 四流：指 RGB、深度、强度、点云全部开启

算法于部分流属于绑定关系，绑定关系如下：

- ❖ 开启托盘算法则必须开启深度
- ❖ 开启避障算法则必须开启深度
- ❖ 开启定位算法则必须开启 RGB

表格中浅蓝色背景为开启点云的运行配置，无色背景为不开启点云的运行配置。以下所有测试数据默认为算法下移，因此对于 **S2** 系列相机来说，所有测试数值可能偏小【**S2 系列相机部分算法强制上移**】。

Ubuntu16.04

ROS1		
运行配置	均值%	区间%
单流	0.60	0.6~0.7
双流	0.61	0.6~0.8
三流	0.66	0.6~0.8
四流	0.79	0.7~0.9
点云+深度	0.60	0.5~0.7
算法+深度	0.50	0.5~0.5
算法+深度+点云	0.60	0.5~0.7
算法+深度+RGB	0.62	0.6~0.8
算法+深度+RGB+点云	0.71	0.6~0.9
算法+深度+强度+RGB+点云	0.79	0.7~0.9

Ubuntu18.04

ROS1		
运行配置	均值%	区间%
单流	1.01	1.1~1.2
双流	1.10	1.1~1.2
三流	1.10	1.1~1.2
四流	1.19	1.1~1.3
点云+深度	1.09	0.9~1.2
算法+深度	0.90	0.9~1.0
算法+深度+点云	1.09	0.9~1.2
算法+深度+RGB	1.10	1.1~1.2
算法+深度+RGB+点云	1.19	1.1~1.3
算法+深度+强度+RGB+点云	1.20	1.1~1.4

ROS2		
运行配置	均值%	区间%
单流	1.19	1.1~1.2
双流	1.27	1.1~1.3
三流	1.29	1.2~1.3
四流	1.57	1.3~1.6
点云+深度	1.20	1.1~1.3
算法+深度	0.90	0.9~0.9
算法+深度+点云	1.10	1.0~1.2
算法+深度+RGB	1.10	1.1~1.2
算法+深度+RGB+点云	1.30	1.3~1.4
算法+深度+强度+RGB+点云	1.41	1.3~1.6

Ubuntu20.04

ROS1		
运行配置	均值%	区间%
单流	1.21	1.2~1.4
双流	1.30	1.3~1.4
三流	1.30	1.3~1.4
四流	1.33	1.3~1.5
点云+深度	1.21	1.2~1.4
算法+深度	1.19	1.1~1.2
算法+深度+点云	1.22	1.2~1.4
算法+深度+RGB	1.30	1.3~1.4
算法+深度+RGB+点云	1.32	1.3~1.5
算法+深度+强度+RGB+点云	1.33	1.3~1.5

ROS2		
运行配置	均值%	区间%
单流	1.20	1.2~1.3
双流	1.20	1.2~1.3
三流	1.30	1.3~1.4
四流	1.64	1.4~1.7
点云+深度	1.36	1.1~1.4
算法+深度	1.10	1.1~1.1
算法+深度+点云	1.19	1.1~1.3
算法+深度+RGB	1.40	1.4~1.4
算法+深度+RGB+点云	1.58	1.4~1.7
算法+深度+强度+RGB+点云	1.60	1.5~1.7

Ubuntu22.04

ROS1		
运行配置	均值%	区间%
单流	1.28	1.1~1.3
双流	1.29	1.2~1.
三流	1.20	1.2~1.3
四流	1.49	1.4~1.6
点云+深度	1.11	1.1~1.2
算法+深度	1.00	1.0~1.0
算法+深度+点云	1.11	1.1~1.2
算法+深度+RGB	1.20	1.2~1.3
算法+深度+RGB+点云	1.55	1.3~1.6
算法+深度+强度+RGB+点云	1.51	1.3~1.7

附录：

本附录中提供一些工具的安装方法，使用其中方法时，至少需要有一定的 linux 基础知识，而且不保证每台电脑都能通过本文档附录中展示的方法成功安装，每台电脑的环境不一样，遇到的问题可能也不一致，如果遇到本附录中没有展示的问题，请自行搜索解决办法。

1. Cmake 指定版本安装方法

前往官网下载指定版本 cmake: <https://cmake.org/download/>

解压下载的包: `tar -zxvf cmake-XXX.tar.gz`

进入解压后的目录，执行编译和配置：

`./bootstrap`

`make`

`sudo make install`

查看 Cmake 版本查看是否安装成功：

`cmake --version`

2. Opencv 指定版本安装方法

❖ 下载所需版本的 opencv: [Releases - OpenCV](#)



❖ 环境配置，依次执行以下命令：

`sudo apt-get install build-essential`

`sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev`

`sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev`

❖ 新建 opencv 文件夹，将下载的源码解压到 opencv 文件夹中

❖ 依次执行以下命令：

```
cd opencv
mkdir build && cd build
sudo cmake -D CMAKE_BUILD_TYPE=Release -D
CMAKE_INSTALL_PREFIX=/usr/local ..
sudo make -j4
sudo make install
```

等待安装完成即可

3. Pcl 指定版本安装方法

查看支持的 pcl 版本：

```
apt-cache madison libpcl-dev
```

安装指定的 pcl 版本（其中 1.10.0 需要替换为所需的版本号）：

```
sudo apt-get install libpcl-dev=1.10.0
```

4. 安装 ROS

Ubuntu16.04

添加软件源，设置密钥：

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

添加密钥：

```
curl -sSL
'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xC1CF6E31E6BADE8868
B172B4F42ED6FBAB17C654' | sudo apt-key add -
```

更新源：

```
sudo apt-get update
```

安装完整的 ros：

```
sudo apt-get install ros-kinetic-desktop-full
```

配置环境变量：

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

补全 ROS 相关的工具：

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

到此 ROS1 全部安装完成，运行一下 ROS 自带的示例，查看安装是否正确：

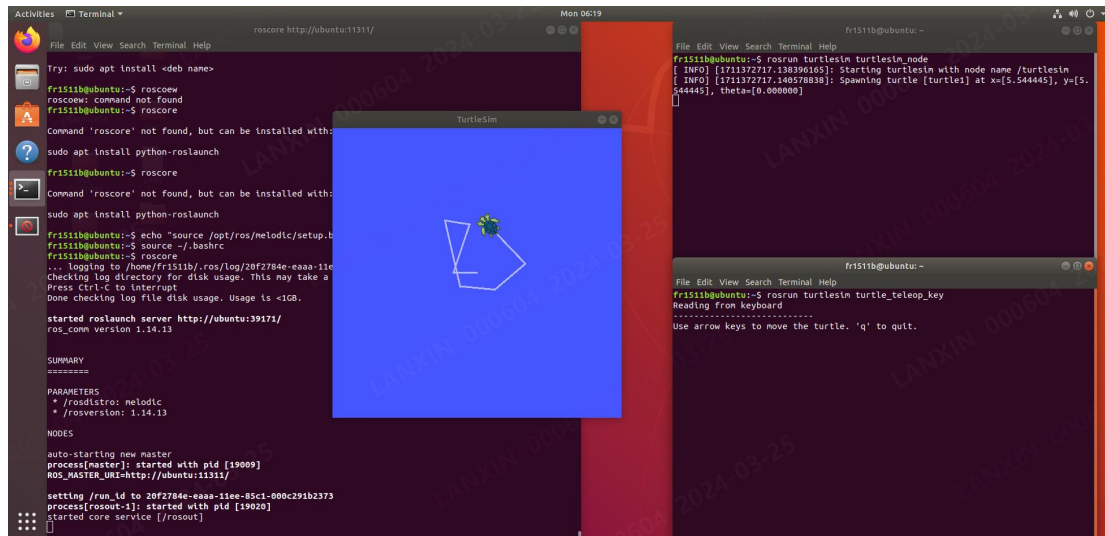
打开一个终端输入：`roscore`

再打开一个终端输入：`roslaunch turtlesim turtlesim_node`

再打开一个终端：`roslaunch turtlesim turtle_teleop_key`

在第三个终端可以通过方向键控制小海龟

如果能正常运行如下图，则说明 ros 已经正确安装：



Ubuntu18.04

设置软件源：

```
sudo sh -c ' . /etc/lsb-release && echo "deb http://mirrors.ustc.edu.cn/ros/ubuntu/`lsb_release -cs` main" > /etc/apt/sources.list.d/ros-latest.list'
```

设置公钥：

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

更新软件列表：

```
sudo apt update
```

安装 Ubuntu18.04 版本的 ROS1：

```
sudo apt install ros-melodic-desktop-full
```

设置环境变量：

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

如果是 zsh：

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.zshrc
```

```
source ~/.zshrc
```

补全 ROS 相关的工具：

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

到此 ROS1 全部安装完成，运行一下 ROS 自带的示例，查看安装是否正确：

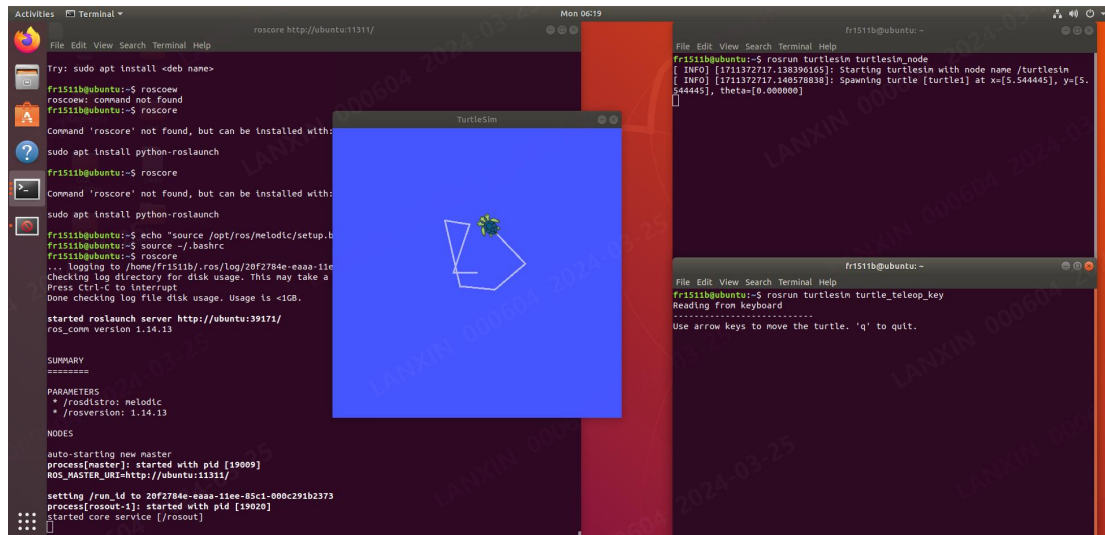
打开一个终端输入：`roscore`

再打开一个终端输入：`roslaunch turtlesim turtlesim_node`

再打开一个终端：`roslaunch turtlesim turtle_teleop_key`

在第三个终端可以通过方向键控制小海龟

如果能正常运行如下图，则说明 ros 已经正确安装：



Ubuntu20.04

配置公钥：

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

如果显示无法连接服务器或超时，请先更新软件源列表：

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

配置公钥后，再次更新软件列表：

```
sudo apt-get update
```

安装 ros：

```
sudo apt install ros-noetic-desktop-full
```

设置环境变量：

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

安装 ROS 工具：

```
sudo apt install python3-rosinstall python3-rosinstall-generator python3-wstool
build-essential
```

到此 ROS1 全部安装完成，运行一下 ROS 自带的示例，查看安装是否正确：

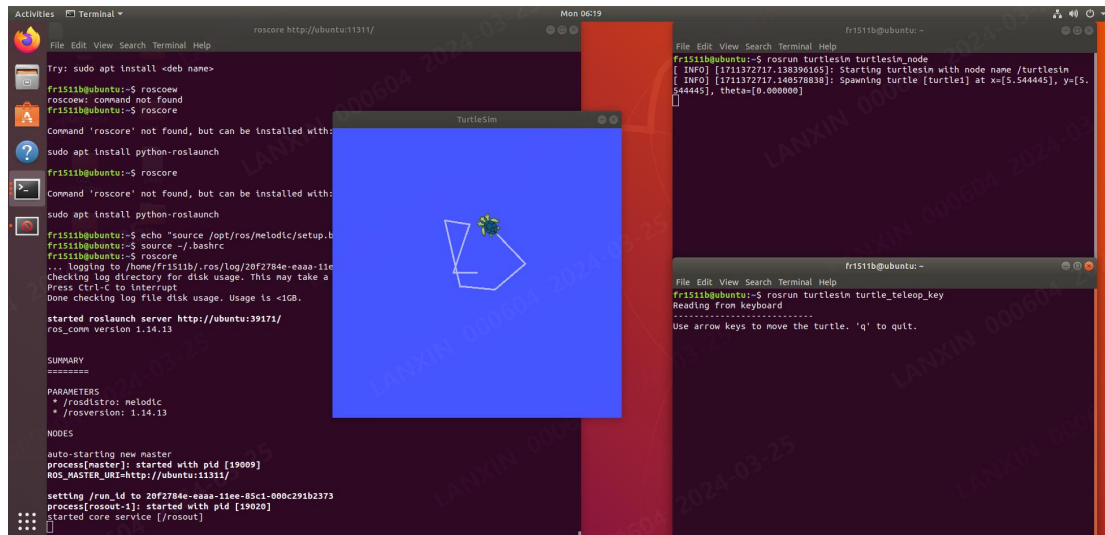
打开一个终端输入：`roscore`

再打开一个终端输入：`roslaunch turtlesim turtlesim_node`

再打开一个终端：`roslaunch turtlesim turtle_teleop_key`

在第三个终端可以通过方向键控制小海龟

如果能正常运行如下图，则说明 ros 已经正确安装：



Ubuntu22.04

不推荐在 Ubuntu22.04 中安装 ros1, 官方并不支持, 如果要安装会遇到非常多的依赖问题。

5. Ros 如何创建一个新的测试包

新建一个文件夹 ws, 在 ws 文件夹中新建 src 文件夹, 进入 src 文件夹执行:

```
catkin_create_pkg example std_msgs rospy roscpp
```

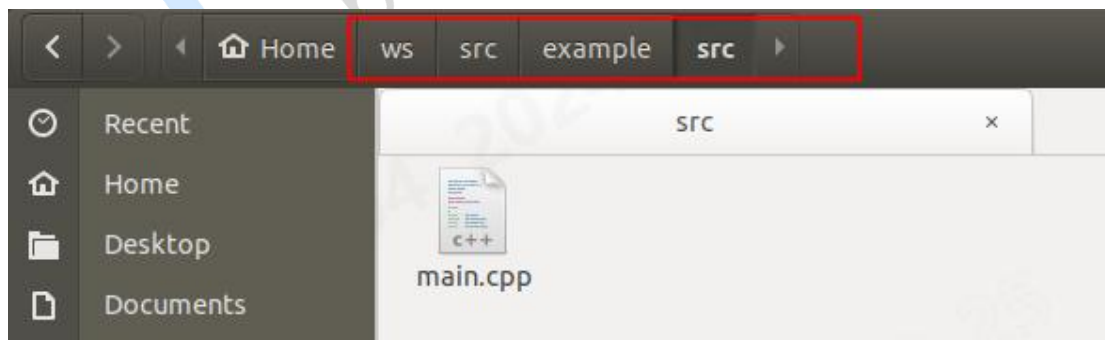
这样就新建了一个名为 example 的包, 包依赖于 std_msgs rospy roscpp

然后编译这个新建包: `catkin_make`

这是一个空的包, 不会有任何可执行程序, 它只是一个包

接下来演示如何创建一个可以输出'Hellow world'的包

在刚刚那个包的基础上, 我们进入 src 文件夹(包里的 src), 并新建一个 main.cpp, 如下图 (注意看路径):



在 main.cpp 中, 我们输入一些代码:

```

Open ▾  main.cpp
~/ws/src/example/src

#include<iostream>

int main(){
    std::cout<<"Hellow world!"<<std::endl;
}

```

然后我们改一下外面的 CMakeLists.txt，我们在最底下加入一行：

`add_executable(example src/main.cpp)`

```

Open ▾  CMakeLists.txt
~/ws/src/example

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_example.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
add_executable(example src/main.cpp)

```

然后返回到 ws 目录中编译，执行命令：`catkin_make`

```

Scanning dependencies of target example
[ 50%] Building CXX object example/CMakeFiles/example.dir/src/main.cpp.o
[100%] Linking CXX executable /home/fr1511b/ws/devel/lib/example/example
[100%] Built target example
fr1511b@ubuntu:~/ws$

```

看到这些，就说明编译成功了

接下来我们配置环境：

`source devel/setup.bash`

```

[ 50%] Building CXX object example/CMakeFiles/example.dir/src/main.cpp.o
[100%] Linking CXX executable /home/fr1511b/ws/devel/lib/example/example
[100%] Built target example
fr1511b@ubuntu:~/ws$ source devel/setup.bash
fr1511b@ubuntu:~/ws$

```

然后执行程序，就可以看到输出了：

`roslaunch example example`

```

fr1511b@ubuntu:~/ws$ source devel/setup.bash
fr1511b@ubuntu:~/ws$ roslaunch example example
Hellow world!
fr1511b@ubuntu:~/ws$

```

6. 安装 ROS2

Ubuntu16.04

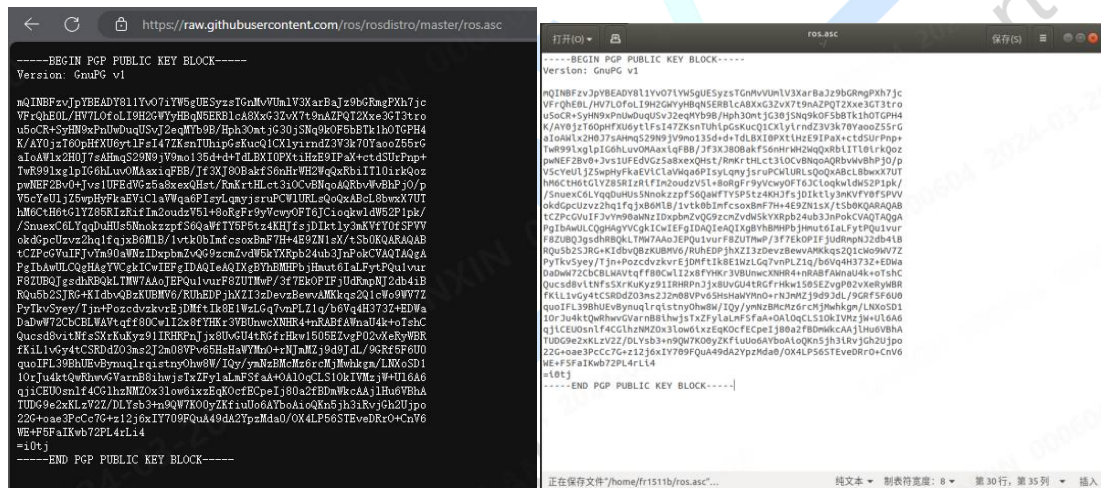
添加密钥：

```
sudo apt-get update && sudo apt-get install curl
curl http://repo.ros2.org/repos.key | sudo apt-key add -
```

添加权限：

```
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 -
```

如果此步骤报错，则使用 VPN 翻墙然后打开浏览器，输入网址，并复制其中内容，然后新建 ros.asc，将内容拷贝到文件中，拷贝完成后如下：



然后重新执行第三步：sudo apt-key add ros.asc

继续执行：

```
sudo sh -c 'echo "deb [arch=amd64] http://packages.ros.org/ros2/ubuntu bionic main" > /etc/apt/sources.list.d/ros2-latest.list'
```

安装 ROS2：

```
sudo apt update
```

```
sudo apt install ros-ardent-desktop
```

配制环境变量：

```
source /opt/ros/ardent/setup.bash
```

安装功能包：

```
sudo apt-get update
```

```
sudo apt-get install ros-ardent-ros1-bridge
```

```
sudo apt-get install ros-ardent-turtlebot2-*
```

尝试运行示例程序，打开一个终端运行：

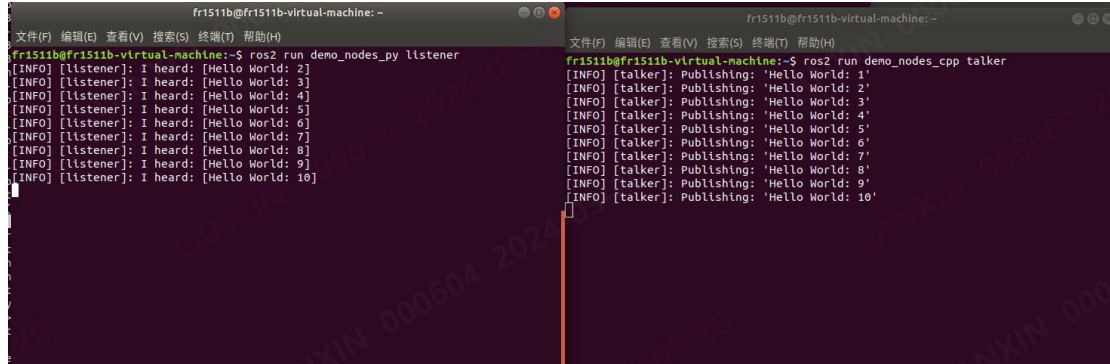
```
source ~/.bashrc
```

```
ros2 run demo_nodes_cpp talker
```

再打开一个中断运行：

Linux 示例程序使用说明

```
source ~/.bashrc
ros2 run demo_nodes_py listener
终端输出如下则证明安装成功:
```



The image shows two terminal windows. The left window shows the output of `ros2 run demo_nodes_py listener`, displaying a series of messages from the listener node: `[INFO] [listener]: I heard: [Hello World: 2]` through `[INFO] [listener]: I heard: [Hello World: 10]`. The right window shows the output of `ros2 run demo_nodes_cpp talker`, displaying a series of messages from the talker node: `[INFO] [talker]: Publishing: 'Hello World: 1'` through `[INFO] [talker]: Publishing: 'Hello World: 10'`.

Ubuntu18.04:

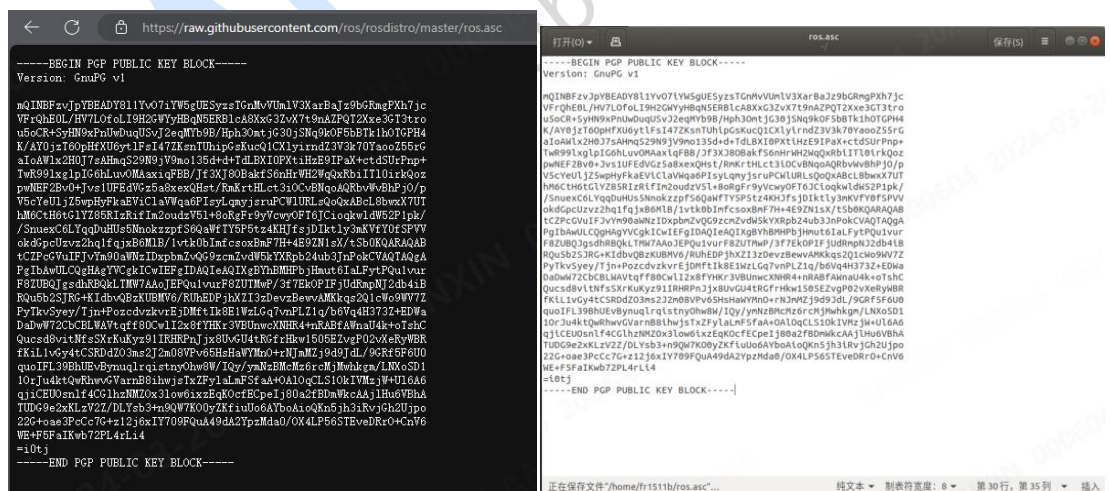
添加软件源

```
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 ros.asc
```

如果此步骤报错如下:

```
fr1511b@fr1511b-virtual-machine:~$ sudo apt-key add ros.asc
gpg: can't open 'ros.asc': 没有那个文件或目录
```

则使用 VPN 翻墙然后打开浏览器, 输入网址, 并复制其中内容, 然后新建 `ros.asc`, 将内容拷贝到文件中, 拷贝完成后如下:



The image shows a web browser window displaying the ROS2 public key from `https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc`. The key is a long string of text starting with `-----BEGIN PGP PUBLIC KEY BLOCK-----`. To the right, a terminal window shows the command `curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc > ros.asc` being executed, and the output `Version: GnuPG v1` is displayed.

然后重新执行第三步: `sudo apt-key add ros.asc`
继续执行:

```
sudo sh -c 'echo "deb [arch=amd64] http://packages.ros.org/ros2/ubuntu bionic
main" > /etc/apt/sources.list.d/ros2-latest.list'
完成后开始下载 ros2, 其中红色字体需要替换为不同的版本 (查看上表):
sudo apt update
```

```
sudo apt install ros-eloquent-desktop
```

等待安装完成后继续安装 python 库，执行：

```
sudo apt install -y libpython3-dev python3-pip
```

```
pip3 install -U argcomplete
```

接着配置环境变量：

```
echo "source /opt/ros/eloquent/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

尝试运行示例程序，打开一个终端运行：

```
source ~/.bashrc
```

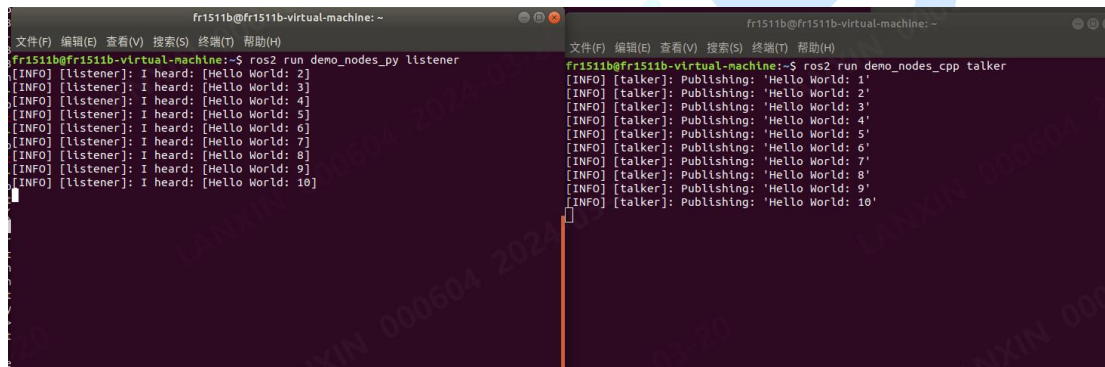
```
ros2 run demo_nodes_cpp talker
```

再打开一个中断运行：

```
source ~/.bashrc
```

```
ros2 run demo_nodes_py listener
```

终端输出如下则证明安装成功：



```
fr1511b@fr1511b-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
fr1511b@fr1511b-virtual-machine:~$ ros2 run demo_nodes_py listener
[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]

fr1511b@fr1511b-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
fr1511b@fr1511b-virtual-machine:~$ 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'
```

Ubuntu20.04

添加软件源

```
sudo apt update && sudo apt install curl gnupg2 lsb-release
```

```
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
```

如果遇到报错“ailed to connect to raw.githubusercontent.com port 443 after 13 ms: 拒绝连接”

```
mier@fr1511b:~$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
curl: (7) Failed to connect to raw.githubusercontent.com port 443 after 0 ms: Connection refused
```

```
sudo vi /etc/hosts
```

增加下面的解析

```
185.199.108.133 raw.githubusercontent.com
```

添加后的文件如下：

```

mier@fr1511b:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      fr1511b
185.199.108.133 raw.githubusercontent.com

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters

```

```

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
继续执行:

```

```

sudo sh -c 'echo "deb [arch=amd64] http://packages.ros.org/ros2/ubuntu bionic
main" > /etc/apt/sources.list.d/ros2-latest.list'

```

完成后开始下载 ros2:

```
sudo apt update
```

```
sudo apt install ros-foxy-desktop
```

等待安装完成后继续安装 python 库, 执行:

```
sudo apt install -y libpython3-dev python3-pip
```

```
pip3 install -U argcomplete
```

接着配置环境变量:

```
echo "source /opt/ros/foxy/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

尝试运行示例程序, 打开一个终端运行:

```
source ~/.bashrc
```

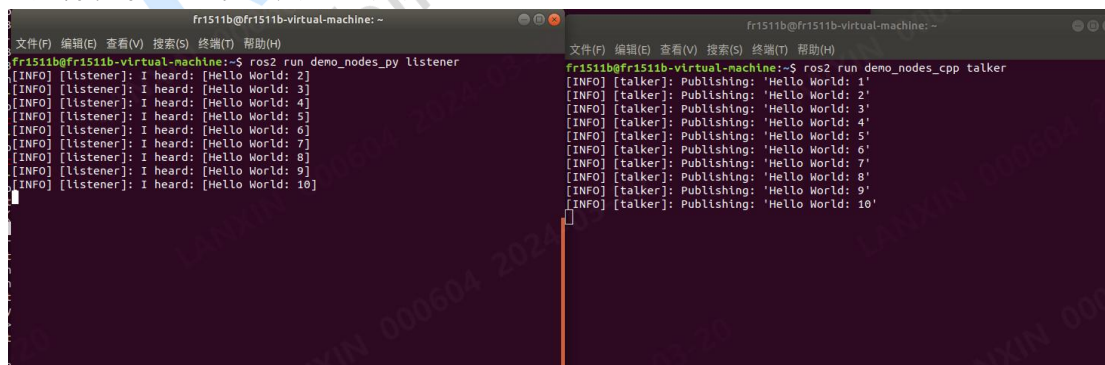
```
ros2 run demo_nodes_cpp talker
```

再打开一个中断运行:

```
source ~/.bashrc
```

```
ros2 run demo_nodes_py listener
```

终端输出如下则证明安装成功:



```

fr1511b@fr1511b-virtual-machine:~$ ros2 run demo_nodes_py listener
[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]

fr1511b@fr1511b-virtual-machine:~$ 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'

```

Ubuntu22.04

添加软件源

```
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
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
如果遇到报错“ailed to connect to raw.githubusercontent.com port 443 after 13 ms:
拒绝连接”
```

```
mier@fr1511b:~$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archi
ve-keyring.gpg
curl: (7) Failed to connect to raw.githubusercontent.com port 443 after 0 ms: Connection refused
```

```
sudo vi /etc/hosts
```

增加下面的解析

```
185.199.108.133 raw.githubusercontent.com
```

添加后的文件如下：

```
mier@fr1511b:~$ cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 fr1511b
185.199.108.133 raw.githubusercontent.com

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

完成后开始下载 ros2：

```
sudo apt update
```

```
sudo apt install ros-humble-desktop
```

等待安装完成后继续安装 python 库，执行：

```
sudo apt install -y libpython3-dev python3-pip
```

```
pip3 install -U argcomplete
```

接着配置环境变量：

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

尝试运行示例程序，打开一个终端运行：

```
source ~/.bashrc
```

```
ros2 run demo_nodes_cpp talker
```

再打开一个中断运行：

```
source ~/.bashrc
```

```
ros2 run demo_nodes_py listener
```

终端输出如下则证明安装成功：

```
fr1511b@fr1511b-virtual-machine: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
fr1511b@fr1511b-virtual-machine:~$ ros2 run demo_nodes_py listener  
[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]  
  
fr1511b@fr1511b-virtual-machine: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
fr1511b@fr1511b-virtual-machine:~$ 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'
```

7. Ros2 如何创建一个新的测试包

执行命令：

```
mkdir example_ws && cd example_ws
```

```
ros2 pkg create --build-type ament_cmake --node-name example_n example_p
colcon build
```

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

```
ros2 run example_p example_n
```

就能新建一个名为 example_p 的包和一个名为 example_n 的节点,运行后会输出:
hello world example_p package