注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

6.16 深度相机传感器仿真

深度相机（Depth Camera）是一种除了获取图像，还能获取场景中物体与相机之间距离（深度）的传感器。与普通彩色相机不同，它能直接获取三维信息。

彩色相机可以通过图像识别获取物体的像素坐标。

注：深度相机默认前方是 z 轴 所以我们要添加虚拟部件调整方位

（我们并没有实质性添加了一个摄像头 只是用虚拟部件来转动调整坐标系）

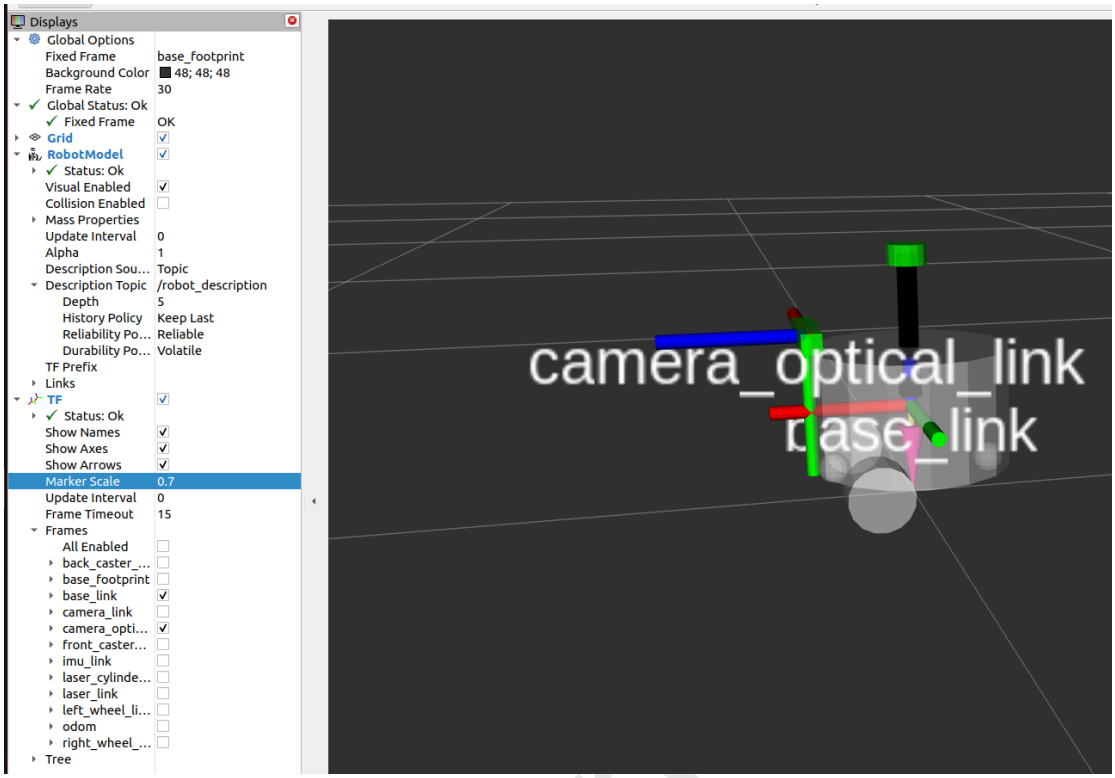
在 camera.urdf.xacro 添加下方代码：

```
<!-- 机器人相机虚拟部件部件 用于调整方位 （深度相机默认前方是z轴） -->
<link name="camera_optical_link"></link>
```

```
<!-- 添加虚拟部件的关节 -->
<joint name="camera_optical_joint" type="fixed">
    <parent link="camera_link"/>
    <child link="camera_optical_link"/>
    <!-- 关节的偏移和旋转量 -->
    <origin xyz="0.0 0.0 0.0" rpy="${-pi/2} 0.0 ${-pi/2}"/>
</joint>
```

打开 rviz 我们可以看到深度相机前方已调整成和 base_link 不一样的坐标系

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可



在 gazebo_sensor_plugin 添加如下代码：

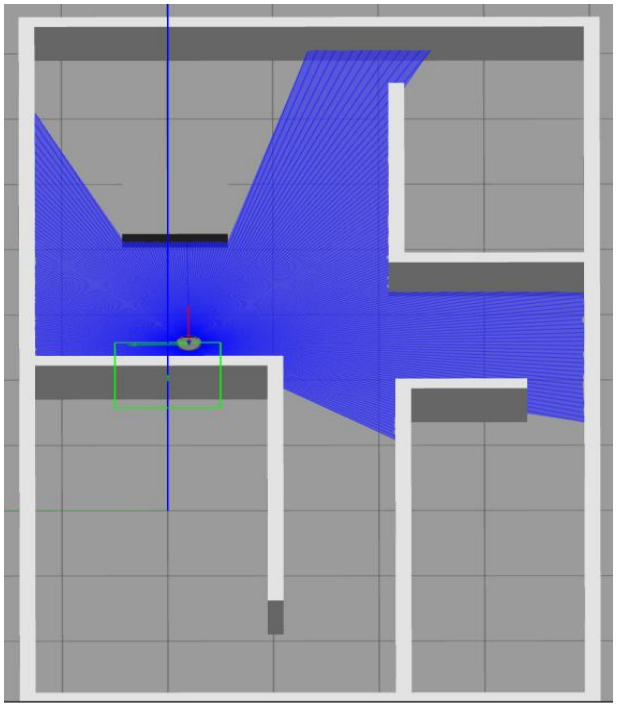
```
<!-- 深度相机传感器插件 （应该安装在虚拟部件上） -->
<gazebo reference="camera_link">
    <sensor name="camera_sensor" type="depth">
        <plugin filename="libgazebo_ros_camera.so" name="depth_camera">
            <frame_name>camera_optical_link</frame_name>
        </plugin>

        <always_on>true</always_on>
        <update_rate>10.0</update_rate>
        <!-- 相机配置 -->
        <camera name="camera">
            <horizontal_fov>1.5009831567</horizontal_fov> <!-- 水平视场角 -->
            <image>
                <width>800</width> <!-- 图像宽度 -->
                <height>600</height> <!-- 图像高度 -->
                <format>R8G8B8</format> <!-- 图像格式 -->
            </image>
            <distortion>
                <k1>0.0</k1> <!-- 径向畸变系数 -->
                <k2>0.0</k2> <!-- 径向畸变系数 -->
                <p1>0.0</p1> <!-- 切向畸变系数 -->
                <p2>0.0</p2> <!-- 切向畸变系数 -->
                <k3>0.0</k3> <!-- 径向畸变系数 -->
                <center>0.5 0.5</center> <!-- 图像中心 -->
            </distortion>
        </camera>
    </sensor>
</gazebo>
```

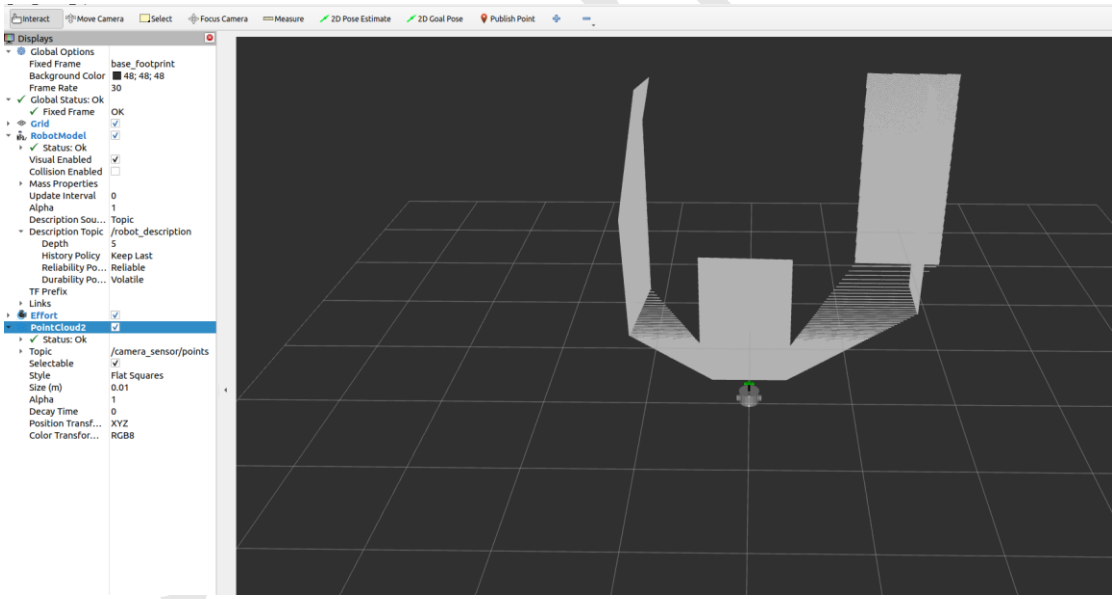
重新构建运行 把机器人拖到以下位置 我们打开 rviz:

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可



在 rviz 中找到 add by topic 选择 PointCloud2 效果显示如下



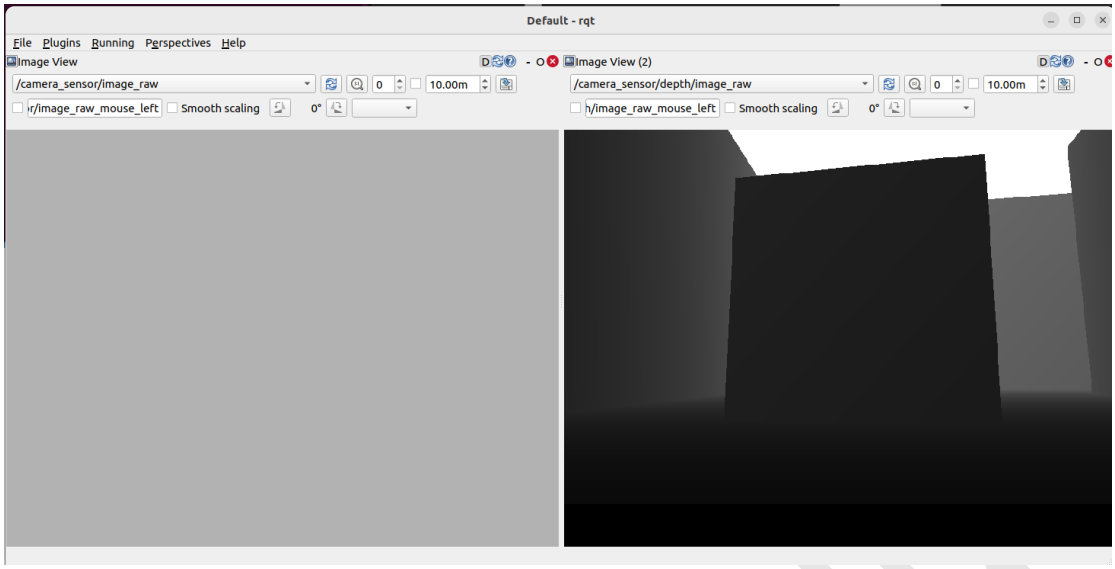
（点云信息）

打开 rqt 在 plugins 里点击最下方 Visualization 选择 Image View

重点看右侧图 图为深度相机 颜色越亮表示距离越远 越深表示越近

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可



到此 我们已经实现了一个基于各种传感器的移动机器人平台的搭建。

但是如果有一些功能 Gazebo 插件无法满足怎么办?

自己搭一个插件？怎么把编写的迁移到真是机器人上？都要重写一遍?

➔ Next Chapter: 通用机器人控制框架 ROS2_Control

四． ROS2_Control 驱动机器人

ros2_control 就像是“机器人控制中心”，它帮你统一管理和控制机器人的马达、电机、舵机这些硬件，不管你用的是仿真环境还是真实机器人。

它相当于一座“电机调度站”，你告诉它“我要电机转动”，它帮你搞定怎么转、转多少、怎么和硬件通信——你不用亲自去接电线、写底层代码。

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

🎮 类比 1：你玩遥控赛车 + 游戏手柄

假设你有一个遥控赛车：

- 前进、后退、左右转，全靠你用遥控器控制；
- 你不需要知道里面电机怎么动、电路怎么连；
- 你只管按方向键，赛车就会动。

➤ 在这里：

- **你就是控制器**（给指令）
- **车里的芯片就是执行器接口**
- **无线遥控协议就是"命令接口"**

如果你换了一台新车，你还是用同样的遥控器——只要通信协议一样，你不需要改操作方式。

这就是 `ros2_control` 的本质：

> 🚗 **"我只管给命令，具体谁去转电机我不管"**
> 💡 控制逻辑独立于具体硬件！这就叫"抽象"。

6.17 ros2_control 介绍与安装

ros2_control 的出现，是为了解决控制器重复造轮子的问题，特别是在像 Gazebo 这样的仿真环境和真实硬件中都需要写重复控制逻辑的场景下。

书中一开始提出：我们常常使用 ROS 控制机器人移动，比如在仿真里用 Gazebo，现实中用电机驱动。

比如你想控制一个**差速小车**（两个轮子，靠左右轮差速转弯），我们通常会这么做：

- 在 Gazebo 里用插件（比如 `libgazebo_ros_diff_drive.so`）控制左右轮；
- 在真机里用电机驱动板自己写代码控制电机。

⚠️ 问题来了：

> 你是不是**写了两遍控制逻辑**？
> 一套给仿真用，一套给真机用？
> → 这就是在"重复造轮子"！

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

🧠 举个类比：遥控小车控制两套系统

你有一个遥控小车：

- 在家里用电脑玩仿真模拟小车转弯；
- 出门用真遥控器控制真小车转弯。

但你每次都要重新学一套控制方式，这不是很累吗？

你希望：

> 不管是在仿真里还是现实中，**我只用同一个遥控器操作，系统自己帮我翻译成轮子的动作。**

这就是 `ros2_control` 要做的事：
→ 统一控制逻辑，接口负责适配不同硬件或仿真

> 既然控制器会重复，那我们就把它抽象出来。

比如说：

- 控制器只管"我想让机器人往前走"；
- 不管你是用 Gazebo 仿真电机，还是现实中用驱动板发 PWM 波控制轮子；
- 控制器统一输出"速度命令"；
- 底层执行交给硬件接口（data interface）完成！

这样：

- 控制器可以复用；
- 你不会再为每一个设备单独写控制逻辑；
- **提升开发效率、减少出错风险。**

Gazebo 两轮差速插件 (libgazebo_ros_diff_drive.so) 工作原理

/cmd_vel 话题被 Gazebo 插件 libgazebo_ros_diff_drive.so 订阅后，插件将线速度与角速度转为左右轮速度，并通过 Gazebo API 写入仿真模型；同时插件从 Gazebo 中读取轮子的状态（角度、速度），并计算机器人的里程计信息（位置与姿态），然后通过 /odom 话题发布给外部节点订阅使用。

这是仿真的控制逻辑不能直接用于实物，所以我们想把 gazebo 和真实硬件都用同一个控制器 -> 同一个数据接口

插件中，控制器对接话题，数据接口对接 gazebo （真实情况对接硬件）

ros2 control frame:

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

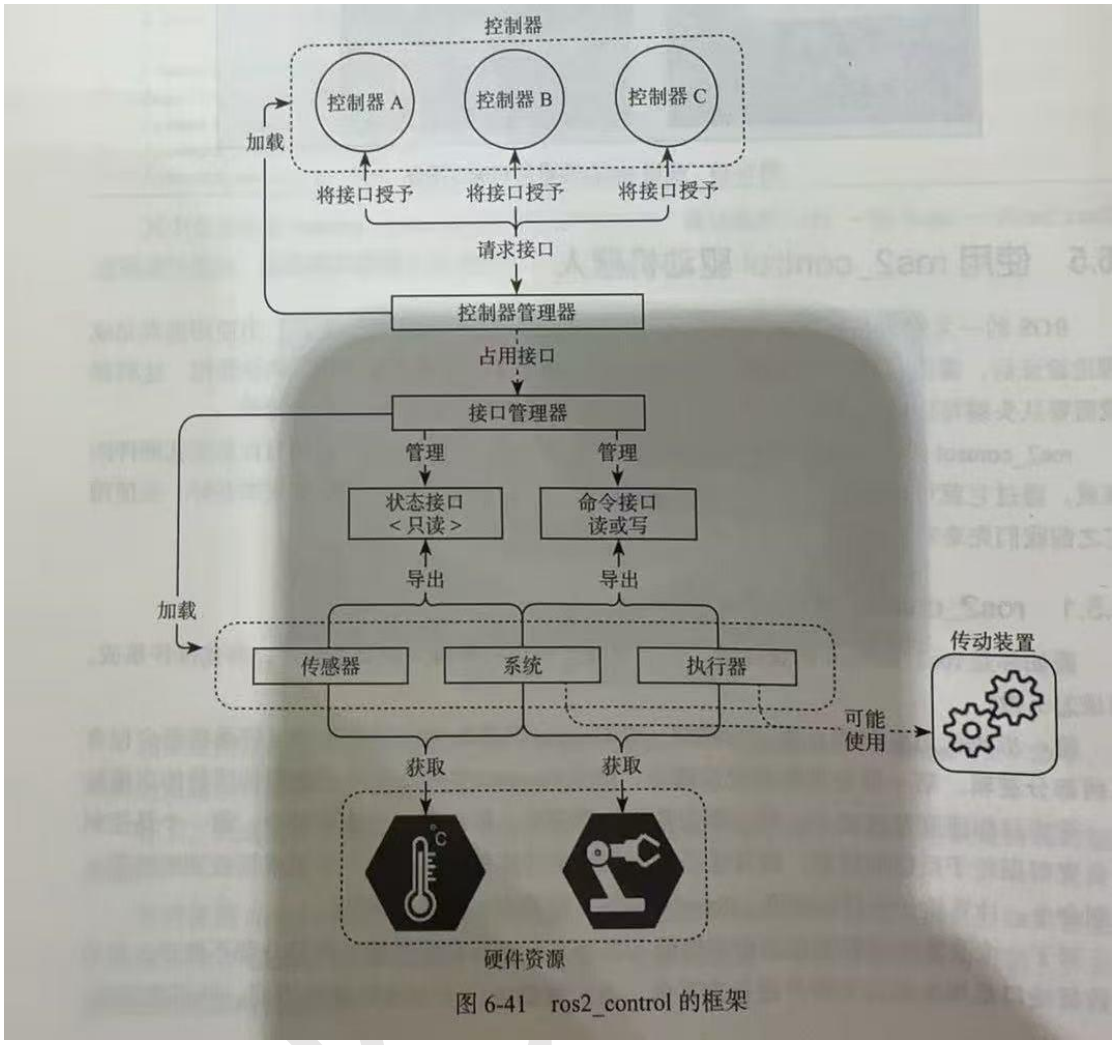


图 6-41　ros2_control 的框架

安装命令： ros2 apt install ros-$ROS_DISTRO-ros2-control

运行下述命令我们可以看到 ros2_control 提供的命令行工具主要针对于硬件接口和控制器的操作。

```
jackson@jackson-virtual-machine:~$ ros2 control --help
usage: ros2 control [-h]
                    Call `ros2 control <command> -h` for more detailed usage.
                    ...

Various control related sub-commands

options:
  -h, --help            show this help message and exit

Commands:
  list_controller_types         Output the available controller types and their
base classes
  list_controllers              Output the list of loaded controllers, their typ
e and status
  list_hardware_components      Output the list of available hardware components
  list_hardware_interfaces      Output the list of available command and state i
nterfaces
  load_controller               Load a controller in a controller manager
  reload_controller_libraries   Reload controller libraries
  set_controller_state          Adjust the state of the controller
  set_hardware_component_state  Adjust the state of the hardware component
  switch_controllers            Switch controllers in a controller manager
  unload_controller             Unload a controller in a controller manager
  view_controller_chains        Generates a diagram of the loaded chained contro
llers into /tmp/controller_diagram.gv.pdf

  Call `ros2 control <command> -h` for more detailed usage.
jackson@jackson-virtual-machine:~$
```

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

但是当我们运行 ros2 control list_controller_types 时会发现报错

这是因为 ros2 control 时基于服务通信的 （Why？）

> 因为**控制器相关的指令**（如加载、启动、停止、切换）都是**一次性、需要明确回应的操作**，这正是**服务通信（Service）最擅长的场景。**

🧠 从“通信语义”出发来解释

我们来分析控制器相关操作的本质：

空制器操作	说明
`load_controller`	加载控制器插件，系统要确认“加载成功了吗”
`configure_controller`	初始化配置资源，要知道“配置成功了吗”
`start_controller`	激活控制器，要知道“是否运行起来”
`switch_controller`	启动一批、停一批，要知道“成功切换了吗”

👉 这些操作的共性：

- 都是 **一次性事务（不是持续数据）**
- 都要 **明确反馈成功或失败**
- 不能丢（命令必须被可靠接收）
- 多个客户端可能发指令，但只允许一个服务端处理

这些特点正好符合 ROS 服务（Service）的通信模型。

```
jackson@jackson-virtual-machine:~$ sudo apt info ros-$ROS_DISTRO-ros2-controller
s
Package: ros-humble-ros2-controllers
Version: 2.47.0-1jammy.20250618.000355
Priority: optional
Section: misc
Maintainer: Bence Magyar <bence.magyar.robotics@gmail.com>
Installed-Size: 44.0 kB
Depends: ros-humble-ackermann-steering-controller, ros-humble-admittance-control
ler, ros-humble-bicycle-steering-controller, ros-humble-diff-drive-controller, r
os-humble-effort-controllers, ros-humble-force-torque-sensor-broadcaster, ros-hu
mble-forward-command-controller, ros-humble-gpio-controllers, ros-humble-gripper
-controllers, ros-humble-imu-sensor-broadcaster, ros-humble-joint-state-broadcas
ter, ros-humble-joint-trajectory-controller, ros-humble-mecanum-drive-controller
, ros-humble-pid-controller, ros-humble-pose-broadcaster, ros-humble-position-co
ntrollers, ros-humble-range-sensor-broadcaster, ros-humble-steering-controllers-
library, ros-humble-tricycle-controller, ros-humble-tricycle-steering-controller
, ros-humble-velocity-controllers, ros-humble-ros-workspace
Homepage: https://control.ros.org
Download-Size: 7,410 B
APT-Manual-Installed: yes
APT-Sources: http://packages.ros.org/ros2/ubuntu jammy/main amd64 Packages
Description: Metapackage for ros2_controllers related packages

N: Ignoring file 'ros2.listsudo apt updatesudo apt install ros-humble-rqt-tf-tre
e -y' in directory '/etc/apt/sources.list.d/' as it has an invalid filename exte
nsion
N: There is 1 additional record. Please use the '-a' switch to see it
```

我们之前用到的 diff-drive-controller / joint_state_broadcaster 这些都是我们之前用到过的控制器 下面我们来安装所有的控制器

sudo apt install ros-$ROS_DISTRO-ros2-controllers

有了这些控制器我们就可以实现精细化控制 因为没有硬件 下面我们将 gazebo 中的仿真硬件接入 ros2_control

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

6.18 使用 Gazebo 接入 ros2_control

使用 Gazebo 接入 ros2_control：让 Gazebo 按照 ros2_control 指定的接口提供数据。在 ROS2 中我们可以方便实现两者对接-> gazebo-ros2-control 插件

sudo apt install ros-$ROS_DISTRO-gazebo-ros2-control

gazebo_ros2_control 对硬件资源的描述同样是 XML，所以在 URDF/jacksonbot 目录下新建 jacksonbot.ros2_control.xacro

键入以下代码：

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:macro name = "jacksonbot_ros2_control">
        <ros2_control name="JacksonbotGazeboSystem" type="system">
            <hardware>
                <plugin>gazebo_ros2_control/GazeboSystem</plugin>
            </hardware>

            <joint name="left_wheel_joint">
                <command_interface name ="velocity">
                    <param name = "min">-1</param>
                    <param name = "max">1</param>
                </command_interface>

                <command_interface name="effort">
                    <param name="min">-0.1</param>
                    <param name="max">0.1</param>
                </command_interface>

                <state_interface name="position"/>
                <state_interface name="velocity"/>
                <state_interface name="effort"/>
            </joint>

            <joint name="right_wheel_joint">
                <command_interface name ="velocity">
                    <param name = "min">-1</param>
                    <param name = "max">1</param>
                </command_interface>

                <command_interface name="effort">
                    <param name="min">-0.1</param>
                    <param name="max">0.1</param>
                </command_interface>

                <state_interface name="position"/>
```

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

注意：本章节有部分代码图片中的注释存在形状体描述错误 但是不影响整体运行 发现问题后改正即可

```
37                  <state_interface name="velocity"/>
38                  <state_interface name="effort"/>
39              </joint>
40          </ros2_control>
41      <gazebo>
42          <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
43              <parameters>$(find firstbot_description)/config/jacksonbot_ros2_controller.yaml</parameters>
44          </plugin>
45      </gazebo>
46      </xacro:macro>
47  </robot>
```

这是一个 xacro 宏定义文件，名字是 jacksonbot_ros2_control，作用是定义机器人（Jacksonbot）的两个车轮关节如何通过 ros2_control 在 Gazebo 中实现控制。

Joint 部分（左右轮）：定义每个关节支持哪些控制接口与状态接口。

（注：关节名称必须和机器人的保持一致，由于要采用 Gazebo 提供数据接口）

提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!