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

command_interface

```xml
<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>
```

- 控制接口 `velocity` 和 `effort` 代表你可以向这个关节发送速度或力矩指令。
- `min/max` 限定输入的控制值范围。

state_interface

```xml
<state_interface name="position"/>
<state_interface name="velocity"/>
<state_interface name="effort"/>
```

- 表示从 Gazebo 模拟中可以获取的位置、速度、力矩状态。

⚠️ 注意：每个关节定义了三个状态接口和两个命令接口。

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[]!

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

插入 Gazebo 插件以启用 ROS 2 控制系统

xml 复制 编辑

```xml
<gazebo>
  <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
    <parameters>$(find firstbot_description)/config/jacksonbot_ros2_controller.yaml</parameters>
  </plugin>
</gazebo>
```

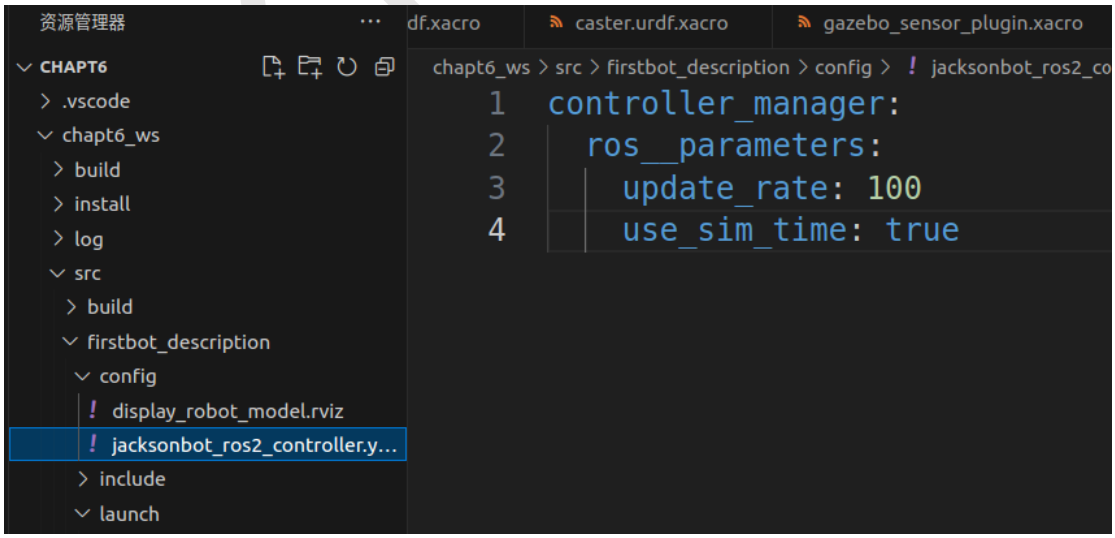
这一段的作用是在 Gazebo 启动时加载 `gazebo_ros2_control` 插件，并使用你提供的控制器参数 YAML 文件。

详细解释：

元素	解释
`<plugin>`	插入 Gazebo 插件，名字为 `gazebo_ros2_control`。
`filename="libgazebo_ros2_control.so"`	这是插件对应的共享库文件，Gazebo 会动态加载这个插件。
`<parameters>`	指定插件启动时加载的控制器配置参数，即你写的 YAML 文件路径。通常放在 `description/config/` 目录下。

这个 YAML 文件（如 `jacksonbot_ros2_controller.yaml`）将会定义控制器的种类、关联的关节名称等，告诉 `controller_manager` 如何管理这些关节。

在 config 目录下新建文件：jacksonbot_ros2_controller.yaml 键入以下代码



在 jacksonbot.urdf.xacro 中删除两轮差速相关宏调用

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

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name='jacksonbot'>

<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/base.urdf.xacro"/>

<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/sensor/laser.urdf.xacro"/>
<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/sensor/camera.urdf.xacro"/>
<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/sensor/imu.urdf.xacro"/>


<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/actuator/wheel.urdf.xacro"/>
<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/actuator/caster.urdf.xacro"/>

<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/plugins/gazebo_control_plugin.xacro"/>
<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/plugins/gazebo_sensor_plugin.xacro"/>

<xacro:include filename="$(find firstbot_description)/urdf/jacksonbot/jacksonbot.ros2_control.xacro"/>

<xacro:base_xacro radius="0.1" length="0.12"/>
<xacro:laser_xacro xyz="0.0 0.0 0.10"/>
<xacro:camera_xacro xyz="0.10 0.0 0.075"/>
<xacro:imu_xacro xyz="0.0 0.0 0.02"/>

<xacro:wheel_xacro wheel_name="left_wheel" xyz="0.05 0.10 -0.06"/>
<xacro:wheel_xacro wheel_name="right_wheel" xyz="0.05 -0.10 -0.06"/>

<xacro:caster_xacro caster_name="front_caster" xyz="0.08 0.0 -0.06"/>
<xacro:caster_xacro caster_name="back_caster" xyz="-0.08 0.0 -0.06"/>

<xacro:gazebo_sensor_plugin/>

<xacro:jacksonbot_ros2_control/>

</robot>
```

运行常见错误：（IMPORTANT！！！）

1. 在运行前，一定要删除所有宏文件（.xacro）中的注释，注意是所有的！
2. Yaml 文件中，ros__parameters 这里是两个下划线
3. 删除掉 jacksonbot.urdf.xacro 中的 gazebo_control_plugin 调用

最终结果：

```
[spawn_entity.py-4] [INFO] [1751347564.746082261] [spawn_entity]: Spawn status: SpawnEntity: Successfully spawned entity [jacksonbot]
[gzserver-2] [INFO] [1751347564.907380941] [gazebo_ros2_control]: Loading gazebo_ros2_control plugin
[gzserver-2] [INFO] [1751347564.996206909] [gazebo_ros2_control]: Starting gazebo_ros2_control plugin in namespace: /
[gzserver-2] [INFO] [1751347564.996534408] [gazebo_ros2_control]: Starting gazebo_ros2_control plugin in ros 2 node: gazebo_ros2_control
[INFO] [spawn_entity.py-4]: process has finished cleanly [pid 15178]
[gzserver-2] [INFO] [1751347565.056673656] [gazebo_ros2_control]: connected to service!! robot_state_publisher
[gzserver-2] [INFO] [1751347565.064745335] [gazebo_ros2_control]: Received urdf from param server, parsing...
[gzserver-2] [INFO] [1751347565.065223034] [gazebo_ros2_control]: Loading parameter files /home/jackson/chapt6/chapt6_ws/install/firstbot_description/share/firstbot_description/config/j
acksonbot_ros2_controller.yaml
[gzserver-2] [INFO] [1751347565.198613898] [gazebo_ros2_control]: Loading joint: left_wheel_joint
[gzserver-2] [INFO] [1751347565.199099397] [gazebo_ros2_control]:       State:
[gzserver-2] [INFO] [1751347565.199312696] [gazebo_ros2_control]:               position
[gzserver-2] [INFO] [1751347565.199520296] [gazebo_ros2_control]:               velocity
[gzserver-2] [INFO] [1751347565.199721995] [gazebo_ros2_control]:               effort
[gzserver-2] [INFO] [1751347565.200136694] [gazebo_ros2_control]:       Command:
[gzserver-2] [INFO] [1751347565.202187289] [gazebo_ros2_control]:               velocity
[gzserver-2] [INFO] [1751347565.205135381] [gazebo_ros2_control]:               effort
[gzserver-2] [INFO] [1751347565.205357681] [gazebo_ros2_control]: Loading joint: right_wheel_joint
[gzserver-2] [INFO] [1751347565.205578380] [gazebo_ros2_control]:       State:
[gzserver-2] [INFO] [1751347565.205989579] [gazebo_ros2_control]:               position
[gzserver-2] [INFO] [1751347565.206191379] [gazebo_ros2_control]:               velocity
[gzserver-2] [INFO] [1751347565.206391878] [gazebo_ros2_control]:               effort
[gzserver-2] [INFO] [1751347565.206593278] [gazebo_ros2_control]:       Command:
[gzserver-2] [INFO] [1751347565.206970777] [gazebo_ros2_control]:               velocity
[gzserver-2] [INFO] [1751347565.209201471] [gazebo_ros2_control]:               effort
[gzserver-2] [INFO] [1751347565.209578870] [resource_manager]: Initialize hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.211290466] [resource_manager]: Successful initialization of hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.213017362] [resource_manager]: 'configure' hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.213710560] [resource_manager]: Successful 'configure' of hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.214227659] [resource_manager]: 'activate' hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.214359558] [resource_manager]: Successful 'activate' of hardware 'JacksonbotGazeboSystem'
[gzserver-2] [INFO] [1751347565.215040257] [gazebo_ros2_control]: Loading controller_manager
[gzserver-2] [WARN] [1751347565.533462653] [gazebo_ros2_control]:  Desired controller update period (0.01 s) is slower than the gazebo simulation period (0.001 s).
[gzserver-2] [INFO] [1751347565.536072647] [gazebo_ros2_control]: Loaded gazebo_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

```
ackson@jackson-virtual-machine:~$ ros2 service list | grep /controller_manager
controller_manager/configure_controller
controller_manager/describe_parameters
controller_manager/get_parameter_types
controller_manager/get_parameters
controller_manager/list_controller_types
controller_manager/list_controllers
controller_manager/list_hardware_components
controller_manager/list_hardware_interfaces
controller_manager/list_parameters
controller_manager/load_controller
controller_manager/reload_controller_libraries
controller_manager/set_hardware_component_state
controller_manager/set_parameters
controller_manager/set_parameters_atomically
controller_manager/switch_controller
controller_manager/unload_controller
```

这条命令列出了所有与 `/controller_manager` 相关的 ROS 2 服务接口。

🔍 含义和作用

这些服务是 ROS 2 控制系统的核心 API，允许你：

服务名	功能说明
`/configure_controller`	配置一个控制器
`/describe_parameters`, `/get_parameters`, `/set_parameters`	管理参数
`/list_controllers`	显示当前已加载的控制器及其状态
`/list_hardware_components`	显示控制器管理器检测到的硬件组件
`/list_hardware_interfaces`	显示所有硬件接口（command/state）
`/load_controller`	加载控制器插件但不启动
`/switch_controller`	激活或停用控制器
`/unload_controller`	卸载控制器
`/reload_controller_libraries`	重新加载控制器库（动态插件支持）
`/set_hardware_component_state`	手动设置硬件状态，例如 `active`

```
jackson@jackson-virtual-machine:~$ ros2 control list_hardware_interfaces
[INFO] [1751348667.414161614] [_ros2cli_16111]: waiting for service /controller_
manager/list_hardware_interfaces to become available...
command interfaces
        left_wheel_joint/effort [available] [unclaimed]
        left_wheel_joint/velocity [available] [unclaimed]
        right_wheel_joint/effort [available] [unclaimed]
        right_wheel_joint/velocity [available] [unclaimed]
state interfaces
        left_wheel_joint/effort
        left_wheel_joint/position
        left_wheel_joint/velocity
        right_wheel_joint/effort
        right_wheel_joint/position
        right_wheel_joint/velocity
```

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

这条命令展示了系统中注册的所有 command 和 state 接口。

📌 显示内容解释

🔧 Command Interfaces

接口	状态
`left_wheel_joint/effort`	[available] [unclaimed]
`left_wheel_joint/velocity`	[available] [unclaimed]
`right_wheel_joint/effort`	[available] [unclaimed]
`right_wheel_joint/velocity`	[available] [unclaimed]

说明你在 URDF 中定义的控制接口已经 **成功注册** 并处于 **可用状态**，但还未被任何控制器"claim"（绑定）。

📡 State Interfaces

这些接口是从模拟器中获取的状态反馈值：

bash 复制 编辑

```
left_wheel_joint/effort
left_wheel_joint/position
left_wheel_joint/velocity
...
```

这些数据将通过 `joint_state_broadcaster` 发布出来。

⚠️ 说明

- **[available]**：表示这个接口已经由底层硬件定义提供。
- **[unclaimed]**：表示还没有控制器来使用这个接口 —— 你尚未启动（或绑定）控制器。

```
jackson@jackson-virtual-machine:~$ ros2 control list_hardware_components
Hardware Component 1
        name: JacksonbotGazeboSystem
        type:
        plugin name:
        state: id=3 label=active
        command interfaces
                left_wheel_joint/velocity [available] [unclaimed]
                left_wheel_joint/effort [available] [unclaimed]
                right_wheel_joint/velocity [available] [unclaimed]
                right_wheel_joint/effort [available] [unclaimed]
```

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

这个命令列出了所有检测到的硬件组件。

📌 输出解读

yaml 复制 编辑

```
Hardware Component 1
  name: JacksonbotGazeboSystem
  state: id=3 label=active
```

字段	说明
`name`	对应 URDF 中 `<ros2_control name="JacksonbotGazeboSystem" ...>`
`state=active`	当前硬件组件已经激活，可以供控制器使用
`command interfaces`	同样显示了四个接口 `[available] [unclaimed]`

这意味着：

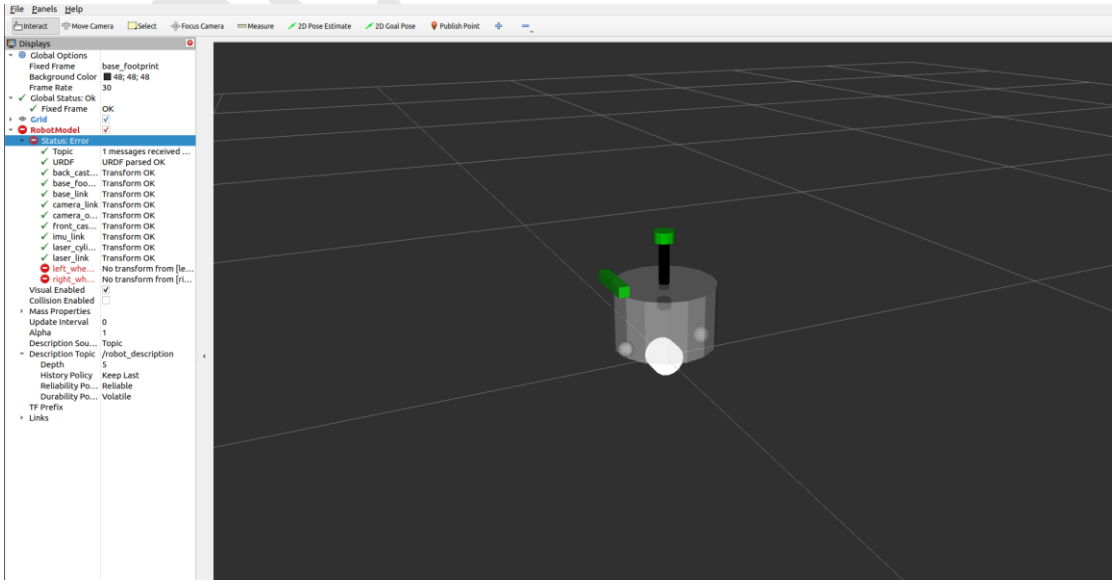
✅ 你的机器人硬件（通过 `gazebo_ros2_control` 插件模拟的）已经 **成功加载并激活。**

这里已经检测到了编号为 1 的组件，一共有 4 个用于控制的命令接口

到这里我们已经实现了把 Gazebo 中仿真机器人的两个轮子接入 ros2_control，接下来我们就可以使用控制器来对接这些接口完成相应功能

6.19 使用关节状态 发布控制器 （发布关节状态的）

上回说到已经把 Gazebo 轮子接入 ROS2_control，但是现在我们打开 Rviz 加载模型发现左侧的 TF 报错，两个轮子的状态位置不对（放在了中间） WHY?



这一现象因为我们删除了两轮差速插件，没有节点提供轮子到 base_footprint

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

的 tf 了 我们可以回去看看之前是谁在发布 打开 gazebo_control_plugin 文件

```
27          <publish_odom>true</publish_odom>
28          <publish_odom_tf>true</publish_odom_tf>
29          <publish_wheel_tf>true</publish_wheel_tf>
30
31          <odometry_frame>odom</odometry_frame>
32          <robot_base_frame>base_footprint</robot_base_frame>
```

通过这段代码我们可以看到是 gazebo_control_plugin 在发布 tf 和里程计关系，那我们现在要解决没有节点发 tf，所以选择使用 ros2_control 的关节状态控制器，发布/joint_states 话题 然后由 robot_state_publisher （这个在 gazebo_sim.launch.py 文件里 所以要保证一直启动这个）订阅转成 TF 再发布

在 config / jacksonbot_ros2_controller.yaml 文件中添加 joint_state_publisher 发布话题

```
6_ws > src > firstbot_description > config > ! jacksonbot_ros2_controller.yaml
1  controller_manager:
2    ros__parameters:
3      update_rate: 100
4      use_sim_time: true
5
6      jacksonbot_joint_state_broadcaster:
7        type: joint_state_broadcaster/JointStateBroadcaster
8
```

给 controlller_manager 添加了一个名称为：jacksonbot_joint_state_broadcaster 的参数表示控制器名称，指定控制器类型: joint_state_broadcaster。该控制器会自动扫描所有状态接口并从中提取 tf 信息并从/joint_state 话题发布。

重新构建运行发现并没有什么变化，还是乱的 -> 因为控制器需要加载并激活！

```
jackson@jackson-virtual-machine:~$ ros2 control
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.
```

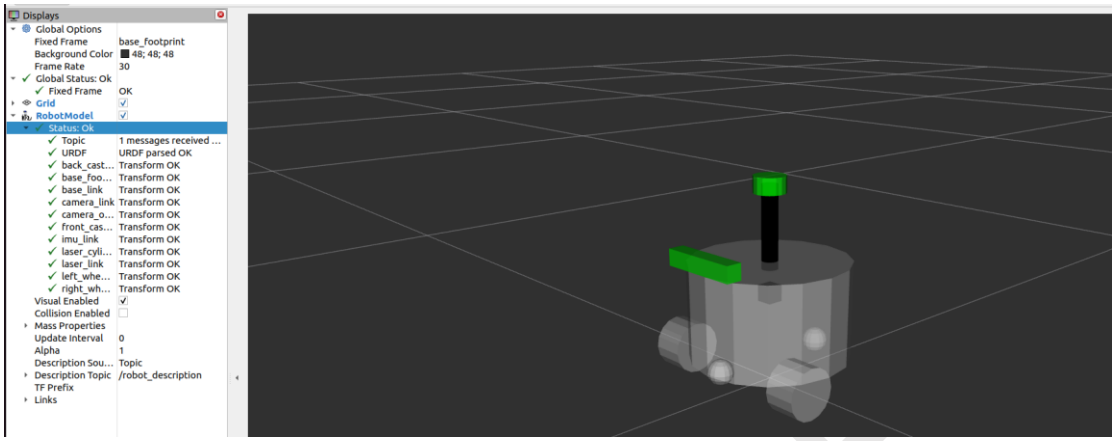
提示：一定要注意各种配置文件代码中的下划线有几个 / 是{}还是[]!

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

可以看到这里面 load_controller 用于加载控制器

```
jackson@jackson-virtual-machine:~$ ros2 control load_controller jacksonbot_joint
_state_broadcaster --set-state active
Successfully loaded controller jacksonbot_joint_state_broadcaster into state act
ive
```

回到 rviz 可以看到已经正常了



通过命令行我们也可以查看/joint_states 话题，显示结果如图：

Ros2 topic echo /joint_states

```
 1.066151836893l3e-05
effort:
 0.0
 0.0
---
header:
 stamp:
   sec: 404
   nanosec: 495000000
 frame_id: ''
name:
 left_wheel_joint
 right_wheel_joint
position:
 0.007997220840853636
 0.023973030105601367
velocity:
 -0.001059127391558286
 0.0006816489528775353
effort:
 0.0
 0.0
---
header:
```

那么每次通过命令行加载控制器太麻烦了 我们统一扔到 launch 文件里

在 gazebo_sim 那个文件在 generate_launch_description 里添加以下代码：

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

```
    # 加载并激活 jacksonbot_joint_state_broadcaster 控制器
    load_joint_state_controller = launch.actions.ExecuteProcess(
        cmd=['ros2', 'control', 'load_controller', '--set-state','active','jacksonbot_joint_state_broadcaster'],
        output='screen'
    )

    return launch.LaunchDescription([
        action_declare_arg_model_path,
        action_robot_state_publisher,
        action_launch_gazebo,
        action_spawn_entity,

        # 事件动作 在机器人加载完成后执行
        launch.actions.RegisterEventHandler(
            event_handler = launch.event_handlers.OnProcessExit(
                target_action=action_spawn_entity,  # 当 action_spawn_entity 执行完成后触发
                on_exit=[load_joint_state_controller]  # 执行 load_joint_state_controller
            )
        )
    ])
```

随后正常启动，并在命令行中键入以下代码，我们可以得到：

查看控制器列表 (已经加载的控制器)

```
jackson@jackson-virtual-machine:~$ ros2 control list_controllers
[INFO] [1751355094.467125559] [_ros2cli_20831]: waiting for service /controller_
manager/list_controllers to become available...
jacksonbot_joint_state_broadcaster joint_state_broadcaster/JointStateBroadcaster
  active
```

也可以卸载控制器　卸载前先转化成未激活状态

```
jackson@jackson-virtual-machine:~$ ros2 control set_controller_state jacksonbot_
joint_state_broadcaster inactive
[INFO] [1751355175.604147135] [_ros2cli_20954]: waiting for service /controller_
manager/list_controllers to become available...
Successfully deactivated jacksonbot_joint_state_broadcaster
```

```
jackson@jackson-virtual-machine:~$ ros2 control unload_controller jacksonbot_joi
nt_state_broadcaster
Successfully unloaded controller jacksonbot_joint_state_broadcaster
```

Ros2 control 可以自由加载卸载控制器，这使得控制器的切换十分灵活，我们可以切换不同的控制器以适应不同需求场景。

下面我们将用控制器来调用控制接口。

6.20 使用力控制器控制轮子

力控的优势：

可以实现柔顺控制

更贴近真实机器人物理行为

在碰撞检测、模拟物理交互、力反馈场景中非常有用。

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 yaml 文件中键入以下代码：

```yaml
controller_manager:
  ros__parameters:
    update_rate: 100
    use_sim_time: true

    jacksonbot_joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster
    jacksonbot_effort_controllers:
      type: effort_controllers/JointGroupEffortController

jacksonbot_effort_controllers:
  ros__parameters:
    joints:
      - front_left_wheel_joint
      - front_right_wheel_joint
      - back_left_wheel_joint
      - back_right_wheel_joint
    command_interface:
      - effort
    state_interface:
      - position
      - velocity
      - effort
```

字段	功能
`joints`	列出该控制器要控制的四个轮子 joint 名称（前左、前右、后左、后右）
`command_interface`	设置为 `"effort"`：你将向这些关节发送力/力矩指令
`state_interface`	读取三个状态接口：位置、速度、力 —— 供反馈或 TF 发布等使用

修改 launch 文件：

```python
# 加载并激活 jacksonbot_effort_controller 控制器
load_jacksonbot_effort_controller = launch.actions.ExecuteProcess(
    cmd=['ros2', 'control', 'load_controller', '--set-state','active','jacksonbot_effort_controller'],
    output='screen'
)
```

```python
# 事件动作 在load_joint_state_controller 执行完成后执行
,launch.actions.RegisterEventHandler(
    event_handler = launch.event_handlers.OnProcessExit(
        target_action=load_joint_state_controller,  # 当 load_joint_state_controller 执行完成后触发
        on_exit=[load_jacksonbot_effort_controller]  # 执行 load_jacksonbot_effort_controllers
    )
)
```

注：launch 文件在启动 Action 时是并行的 控制器激活顺序最好按序依次进行

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

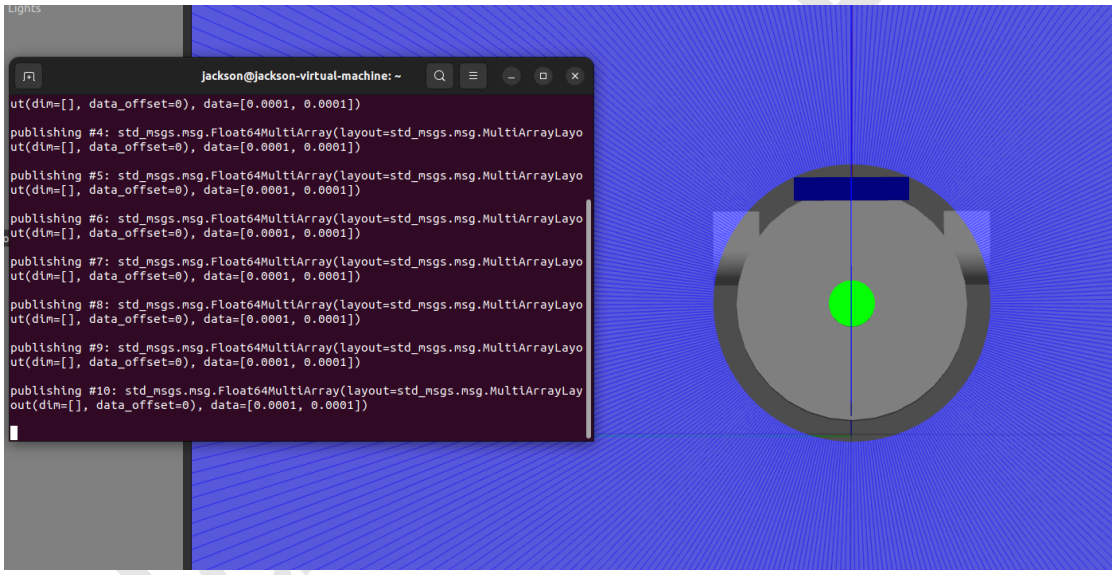
以防止服务接口抢占导致启动异常。可以等 load_joint_state_controller 结束后再运行。

重新构建启动后，命令行运行以下代码可以看到两个话题，其中commands就是用于发布命令的话题

```
rjackson@jackson-virtual-machine:~$ ros2 topic list -v | grep jacksonbot_effort_
ontroller
 * /jacksonbot_effort_controllers/transition_event [lifecycle_msgs/msg/Transitio
nEvent] 1 publisher
 * /jacksonbot_effort_controllers/commands [std_msgs/msg/Float64MultiArray] 1 su
bscriber
jackson@jackson-virtual-machine:~$
```

Ros2 topic pub /jacksonbot_effort_controller/commands std_msgs/msg/Float64MultiArray "{data:[0.0001,0.0001]}"

可以看到机器人在向前缓慢移动并且非常平稳



查看硬件接口发现力相关接口已经被占用

```
jackson@jackson-virtual-machine: ~
ros2 cojackson@jackson-virtual-machine:~$ ros2 control list_hardware_interfaces
command interfaces
        left_wheel_joint/effort [available] [claimed]
        left_wheel_joint/velocity [available] [unclaimed]
        right_wheel_joint/effort [available] [claimed]
        right_wheel_joint/velocity [available] [unclaimed]
state interfaces
        left_wheel_joint/effort
        left_wheel_joint/position
        left_wheel_joint/velocity
        right_wheel_joint/effort
        right_wheel_joint/position
        right_wheel_joint/velocity
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

6.21 使用两轮差速控制器控制机器人

在前面的章节中，我们学习了状态发布控制器和力控制器，它们的作用主要是获取机器人当前的状态信息，并对外提供一个简单的接口。这些控制器做的事情都比较“被动”，只是获取或者转发数据，逻辑比较简单。

但在本节中，我们要学习的是一个更“主动”的控制器：两轮差速控制器

这个控制器的工作远不止是简单地接收和转发信息，它还需要进行运动学计算。具体来说，它会根据目标速度指令，算出机器人左右两个轮子各自需要转多快，然后控制电机让轮子按要求运动。与此同时，它还可以根据轮子的实际运动情况，估算出机器人的当前位置和姿态，这就得到了我们常说的里程计信息。

通俗点说：

状态控制器就像一个传感器读取器，只读数据；

力控制器像一个遥控按钮，控制机器人“用多大力”推东西；

而两轮差速控制器更像一个“老司机”，会根据你告诉它的目标速度，自己计算出左右轮该怎么动，自己还会“猜”自己走到哪里了。

配置 yaml 文件

```
10      jacksonbot_diff_drive_controller:
11        type: diff_drive_controller/DiffDriveController
```

下面这段代码看不懂看下面解释，有关于参数解释可看书 P219 表格或自己上网搜问 GPT 都行

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

```yaml
jacksonbot_diff_drive_controller:
  ros__parameters:
    left_wheel_names: ["left_wheel_joint"]
    right_wheel_names: ["right_wheel_joint"]

    wheel_separation: 0.2
    wheel_radius: 0.032

    wheel_separation_multiplier: 1.0
    left_wheel_radius_multiplier: 1.0
    right_wheel_radius_multiplier: 1.0

    publish_rate: 50.0
    odom_frame_id: odom
    base_frame_id: base_footprint
    pose_covariance_diagonal: [0.001, 0.001, 0.0, 0.0, 0.0, 0.01]
    twist_covariance_diagonal: [0.001, 0.0, 0.0, 0.0, 0.0, 0.01]

    enable_odom_tf: true
    open_loop: true

    cmd_vel_timeout: 0.5
    use_stamped_vel: false
```

校准项（不常改动）

```yaml
wheel_separation_multiplier: 1.0
left_wheel_radius_multiplier: 1.0
right_wheel_radius_multiplier: 1.0
```

- 给你用来调节"实际参数 vs 模型参数"的补偿项
- 一般用于修正轮子打滑、模型偏差等问题
- 如果机器人转向过度/不足，可以适当调节这些乘子

指令超时（解释你之前的问题！）

```yaml
cmd_vel_timeout: 0.5
```

- 重点参数！表示如果 0.5 秒内没接到新的 `/cmd_vel` 指令，控制器就自动停下来
- 所以你之前"按一下只能走一小段"的原因就是这个！
  - 如果 0.5 秒内不再发速度，它就认为"无人控制了"，就停了

继续修改 launch 文件，注意因为 diff_drive 和 effort 都能控制机器人轮子运动，为了防止冲突我们注释掉（或删除）所有有关力控的动作

（不用动 controller_manager）

```python
# 加载并激活 jacksonbot_effort_controller 控制器
load_jacksonbot_diff_drive_controller = launch.actions.ExecuteProcess(
    cmd=['ros2', 'control', 'load_controller', '--set-state','active','jacksonbot_diff_drive_controller'],
     output='screen'
)
```

```python
 ,launch.actions.RegisterEventHandler(
     event_handler = launch.event_handlers.OnProcessExit(
         target_action=load_joint_state_controller,  # 当 load_joint_state_controller 执行完成后触发
         on_exit=[load_jacksonbot_diff_drive_controller]  # 执行 load_jacksonbot_diff_drive_controllers
     )
 )
```

接着在命令行我们可以看到 topic 这里有三个 odom 就是里程计信息，cmd_vel_unstamped 就是前面说的 cmd_vel

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

```
jackson@jackson-virtual-machine:~$ ros2 topic list -v | grep jacksonbot_diff_dri
ve_controller
 * /jacksonbot_diff_drive_controller/odom [nav_msgs/msg/Odometry] 1 publisher
 * /jacksonbot_diff_drive_controller/transition_event [lifecycle_msgs/msg/Transi
tionEvent] 1 publisher
 * /jacksonbot_diff_drive_controller/cmd_vel_unstamped [geometry_msgs/msg/Twist]
 1 subscriber
```

因为控制器所生成的话题都是以控制器名字为前缀的 为了后续操作键盘等我们可以进行映射 打开 xacro 进行如下代码设置

```
<gazebo>
    <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
    <robot_param>robot_description</robot_param>
    <robot_param_node>robot_state_publisher</robot_param_node>
        <parameters>$(find firstbot_description)/config/jacksonbot_ros2_controller.yaml</parameters>

    <ros>
<remapping>/jacksonbot_diff_drive_controller/cmd_vel_unstamped:=/cmd_vel</remapping>
<remapping>/jacksonbot_diff_drive_controller/odom:=/odom</remapping>
    </ros>
    </plugin>
</gazebo>
```

为啥 <robot_param> 和 <parameters> 这些内容现在必须加上，而不是像以前那样只 remapping 一下就能用了?

你用的是这样的配置：

```xml
<plugin name="diff_drive" filename="libgazebo_ros_diff_drive.so">
  <ros>
    <remapping>/cmd_vel:=/my_cmd_vel</remapping>
  </ros>
</plugin>
```

这个插件的本质逻辑：

✅ 它的“聪明”是这样实现的：

- 它**直接从** `<plugin>` **标签里**知道：
  - 哪两个轮子 joint 是左轮和右轮
  - 轮距、半径
  - 订阅哪个话题（默认 `/cmd_vel`）

所以它说：

> “我就控制我自己这个机器人模型里的轮子，反正你加我这个插件，我就服务你。”

也就是说：

- 它就是你模型的一部分
- 它就控制“它附着的机器人”
- 不用你告诉它“机器人是谁”，因为它已经和这个机器人**绑死了**

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

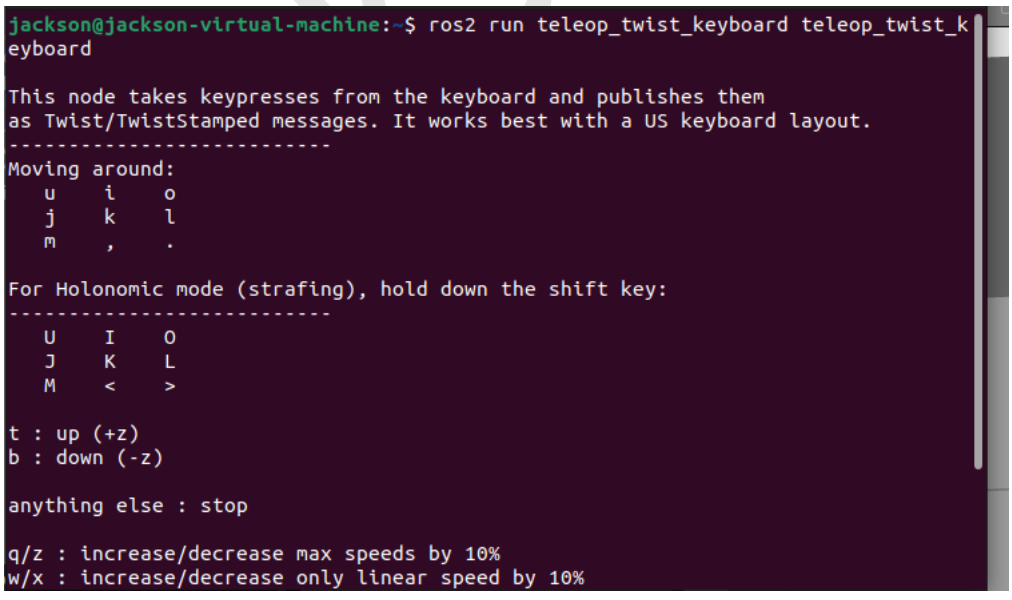
现在你换了个“高级自动驾驶控制系统”：

- 插件是 `gazebo_ros2_control`（它不再是遥控车，而是个控制系统）
- 它不自己直接控制轮子，而是：
  - 启动一个“控制器管理器”（controller_manager）
  - 去找“机器人结构”（`robot_description`，也就是 URDF）
  - 去加载“控制器配置”（比如 `diff_drive_controller` 的参数）
- 然后，才会**根据你发的** `/cmd_vel` 控制机器人轮子

所以你现在需要告诉它：

1. 我的机器人是啥构造（通过 `<robot_param>`）
2. 控制轮子用什么参数（通过 `<parameters>` 加 YAML 文件）
3. 然后你才可以 remap

重新构建仿真，运行之前讲的键盘控制节点就可以操控车了，这里区别于前面的 gazebo_ros_diff_drive，一按前进就一直走，这里是受限于 timeout，只要 timeout 时间内没接收到新命令就停下来，更具有实际意义（防撞防故障）



至此我们已经完成了 ros2_control 结合 Gazebo 学习。

至于怎么把真实硬件接入我们在后续可能会学到 在代码中引入 hardware_interface。

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

章节总结：

在本章中，我们完成了一个典型移动机器人从模型搭建到仿真验证、再到传感器集成与初步控制配置的完整流程，为后续开展机器人自主导航与运动控制系统打下了坚实的基础。

一、机器人建模与仿真环境部署

本章首先介绍了如何使用 URDF/Xacro 宏文件 描述语言对机器人底盘、传感器、执行器等部件进行建模并使用了很多插件。通过 Gazebo 搭建仿真环境，实现了机器人的虚拟运行测试，并结合 RViz 进行模型结构、TF 变换以及传感器数据的可视化，提升了对整体系统结构的直观理解。

二．ROS 2 控制框架基础引入

我们简单介绍了 ros2_control 框架的基本结构与用途，重点演示了如何在 仿真中接入 ros2_control 控制器（状态、力、两轮差速），为后续加载控制插件和集成控制器提供前提。这里并未深入控制器实现细节，而是作为整体系统框架学习的一个过渡。

三．为学习下一张基于 ROS2 导航系统框架 Navigation2 做准备

在接下来的模块中，我们将进一步掌握路径规划、局部避障、目标跟踪等高级功能的实现，并完成从感知到决策再到运动执行的机器人闭环控制体系构建。

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[]!