

26.About ROS2 Gazebo

1.Gazebo introduction

Gazebo is the most commonly used **3D physics simulation platform** in ROS systems. It supports a dynamics engine and can achieve high-quality graphics rendering. It can not only simulate robots and surrounding environments, but also add physical attributes such as friction and elastic coefficients.

For example, if we want to develop a Mars rover, we can simulate the Martian surface environment in Gazebo. Another example is that we are making drones. Due to battery life and flight restrictions, we are unable to frequently conduct experiments with real objects. At this time, we might as well use Gazebo to simulate first, and then deploy the algorithm to real objects for operation after the algorithm is almost developed.

Therefore, simulation platforms like Gazebo can help us verify robot algorithms, optimize robot design, test robot scenario applications, and provide more possibilities for robot development.

Note: This chapter is only for understanding and learning. The environment is not configured in the tutorial because real machine debugging is used directly

2. Installation and operation

Install through the command apt

```
sudo apt install ros-foxy-gazebo-*
```

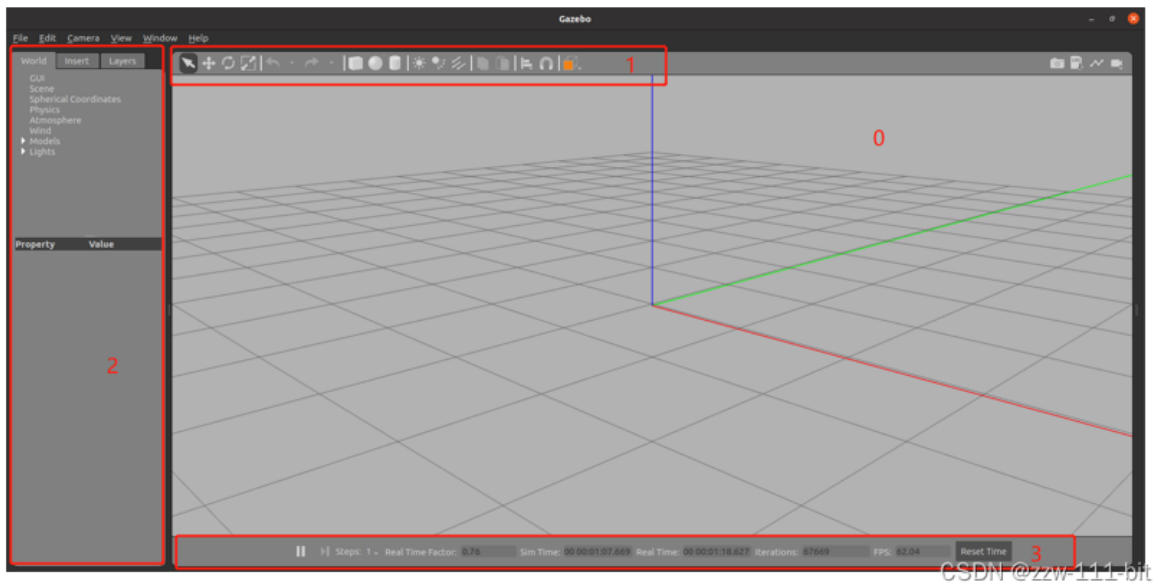
Then you can operation it with the following command

```
ros2 launch gazebo_ros gazebo.launch.py
```

Or the following command

```
gazebo --verbose -s libgazebo_ros_init.so -s libgazebo_ros_factory.so
```

After running, you can see the following page:



To ensure smooth loading of the model, please download and place the offline model in the `~/.gazebo/models` path. The download link is as follows: https://github.com/osrf/gazebo_models

3. Gazebo starts the node and service

1. View nodes

```
ros2 node list
```

Correct return:/gazebo

2. Check the external services provided by the node:

```
ros2 service list
```

You can see the following results:

```
/delete_entity
/get_model_list
/spawn_entity
/gazebo/describe_parameters
/gazebo/get_parameter_types
/gazebo/get_parameters
/gazebo/list_parameters
/gazebo/set_parameters
/gazebo/set_parameters_atomically
```

Out of the last few regular services, we only pay attention to the first three special services:

- `/spawn_entity`, used to load models into gazebo
- `/get_model_list`, used to get the model list
- `/delete_entity`, used to delete models that have been loaded in gazebo

4. Create workspace and function package

Create a gazebo_demo folder, create a src file in it, and then create our function package myrobot in the src folder (take c++ as an example).

```
mkdir -p gazebo_demo/src
cd gazebo_demo/src
ros2 pkg create myrobot --build-type ament_cmake
```

Enter the directory of myrobot, create the launch and urdf folders, and modify the CMakeLists.txt file. This step is similar to that in rviz. Create a demo01_base.urdf file in the urdf folder. This file is a simple demonstration file with only a basic cube.

```
<robot name="myrobot">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.2 0.2 0.2"/>
      </geometry>
      <origin xyz="0.0 0.0 0.0"/>
    </visual>
    <collision>
      <geometry>
        <box size="0.2 0.2 0.2"/>
      </geometry>
      <origin xyz="0.0 0.0 0.0"/>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="0.000190416666667" ixy="0" ixz="0" iyy="0.0001904"
iyy="0" izz="0.00036"/>
    </inertial>
  </link>
  <gazebo reference="base_link">
    <material>Gazebo/Red</material>
  </gazebo>
</robot>
```

The difference between this file and the previous file is that the collision and inertial attributes are added, both of which are physical related parameters. This will not be discussed.

5. Write launch file

When writing a launch file, the launch file mainly starts two parts: starting the Gazebo file, and then loading the robot model into Gazebo.

```
start_gazebo_cmd = ExecuteProcess(
    cmd=['gazebo', '--verbose', '-s', 'libgazebo_ros_init.so', '-s',
'libgazebo_ros_factory.so'],
    output='screen')
```

This command is to start Gazebo. It is just a startup command. It is not particularly complicated. The following is the command to load the model:

```
spawn_entity_cmd = Node(
    package='gazebo_ros',
    executable='spawn_entity.py',
    arguments=['-entity', robot_name_in_model, '-file', urdf_model_path ],
    output='screen')
```

Note that the last two parameters of this command -entity is the name in the model file, and -file is the parameter loaded through the urdf file. Later we can also see that the model is loaded through the topic topic. The complete startup file is as follows:

```
import os
from launch import LaunchDescription
from launch.actions import ExecuteProcess
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
from launch_ros.parameter_descriptions import ParameterValue
from launch.substitutions import Command

def generate_launch_description():
    robot_name_in_model = 'myrobot'
    package_name = 'myrobot'
    urdf_name = "demo01_base.urdf"

    ld = LaunchDescription()
    pkg_share = FindPackageShare(package=package_name).find(package_name)
    urdf_model_path = os.path.join(pkg_share, f'urdf/{urdf_name}')

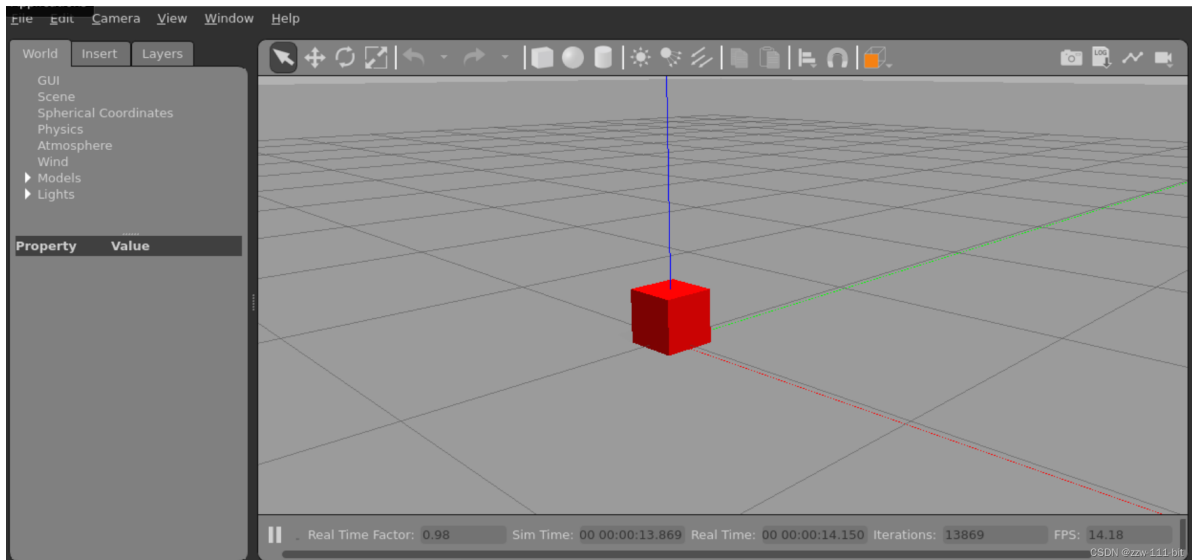
    # Start Gazebo server
    start_gazebo_cmd = ExecuteProcess(
        cmd=['gazebo', '--verbose', '-s', 'libgazebo_ros_init.so', '-s',
            'libgazebo_ros_factory.so'],
        output='screen')

    # Launch the robot
    spawn_entity_cmd = Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        arguments=['-entity', robot_name_in_model, '-file', urdf_model_path ],
        output='screen')

    ld.add_action(start_gazebo_cmd)
    ld.add_action(spawn_entity_cmd)

    return ld
```

After starting, you can see the following Gazebo model:



You can see the red model because the Gazebo label setting was added at the end.

6. Ignition: The next generation of Gazebo

With the advancement of technology, the Gazebo simulation platform is also constantly iterating. The new generation of Gazebo is named **Ignition**. It has major changes in rendering effects and simulation smoothness. We might as well give it a try.

```
$ sudo apt install ros-foxy-ros-ign
$ ros2 launch ros_ign_gazebo_demos rgbd_camera_bridge.launch.py
```

After running successfully, the simulation interface of Ignition and the Rviz host computer will be opened. We can see the image data released after the RGBD camera simulation.

