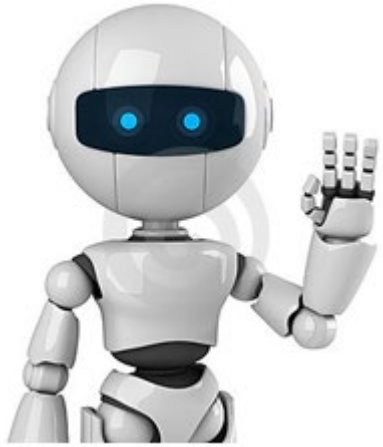
第七章：3D建模与仿真



东北大学 张云洲

2020年6月

学习内容

- 创建机器人的3D模型
- 为机器人提供运动、物理限制、惯性和其他物理响应
- 为机器人3D模型添加仿真传感器
- 在仿真环境中使用该模型

本章提纲

7.1 在ROS中自定义机器人的3D模型

7.2 创建第一个URDF文件

7.3 xacro——一个更好的机器人建模方法

7.4 在ROS中仿真

7.1 在ROS中自定义机器人的3D模型

- 机器人3D模型或部分结构模型主要用于仿真机器人或者为了帮助开发者简化他们的日常工作，在ROS中这通过URDF文件实现。

- 标准化机器人描述格式（URDF）是一种用于描述机器人、其部分结构、关节、自由度等的XML格式文件。

7.2 创建第一个URDF文件

1. 制做一个带有四个轮子的机器人底座

在chapter7_tutorials/robot1_description/urdf文件夹下

创建一个新文件并命名为robot1.urdf，将下面的代码复制到文件中。

```
<?xml version="1.0"?>
  <robot name="Robot1">
    <link name="base_link">
      <visual>
        <geometry>
          <box size="0.2 .3 .1"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0.05"/>
        <material name="white">
          <color rgba="1 1 1 1"/>
        </material>
      </visual>
    </link>

    <link name="wheel_1">
      <visual>
        <geometry>
          <cylinder length="0.05" radius="0.05"/>
        </geometry>
```

2. 解释文件格式

有两种用于描述机器人几何结构的基本字段：连接（link）和关节（joint）

```
<link name="base_link">
  <visual>
    <geometry>
      <box size="0.2 .3 .1" />
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0.05" />
    <material name="white">
      <color rgba="1 1 1 1" />
    </material>
  </visual>
</link>
```

连接的名字，必须唯一

几何形状

材质

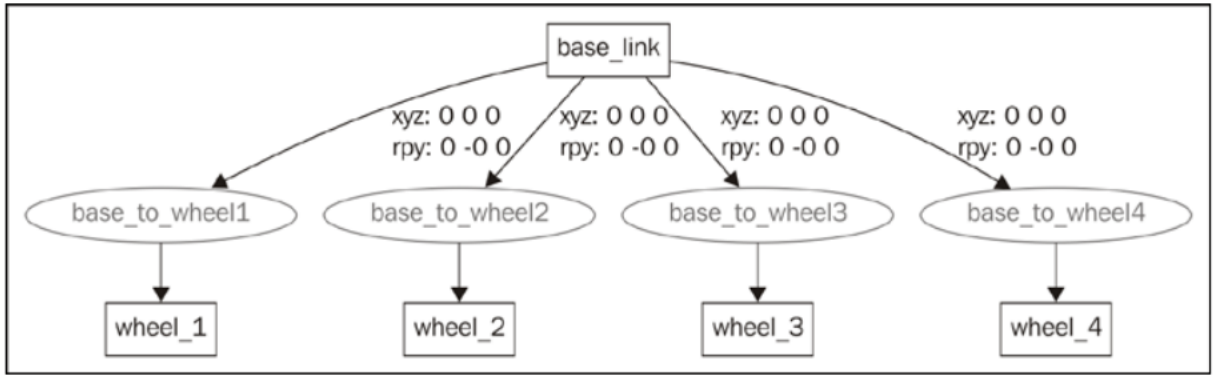
3. 检查文件是否正确配置

$ check_urdf robot1.urdf

```
robot name is: Robot1
---------- Successfully Parsed XML ---------------

root Link: base_link has 4 child(ren)
  child(1):  wheel_1
  child(2):  wheel_2
  child(3):  wheel_3
  child(4):  wheel_4
```

$ urdf_to_graphiz robot1.urdf

$ evince origins.pdf



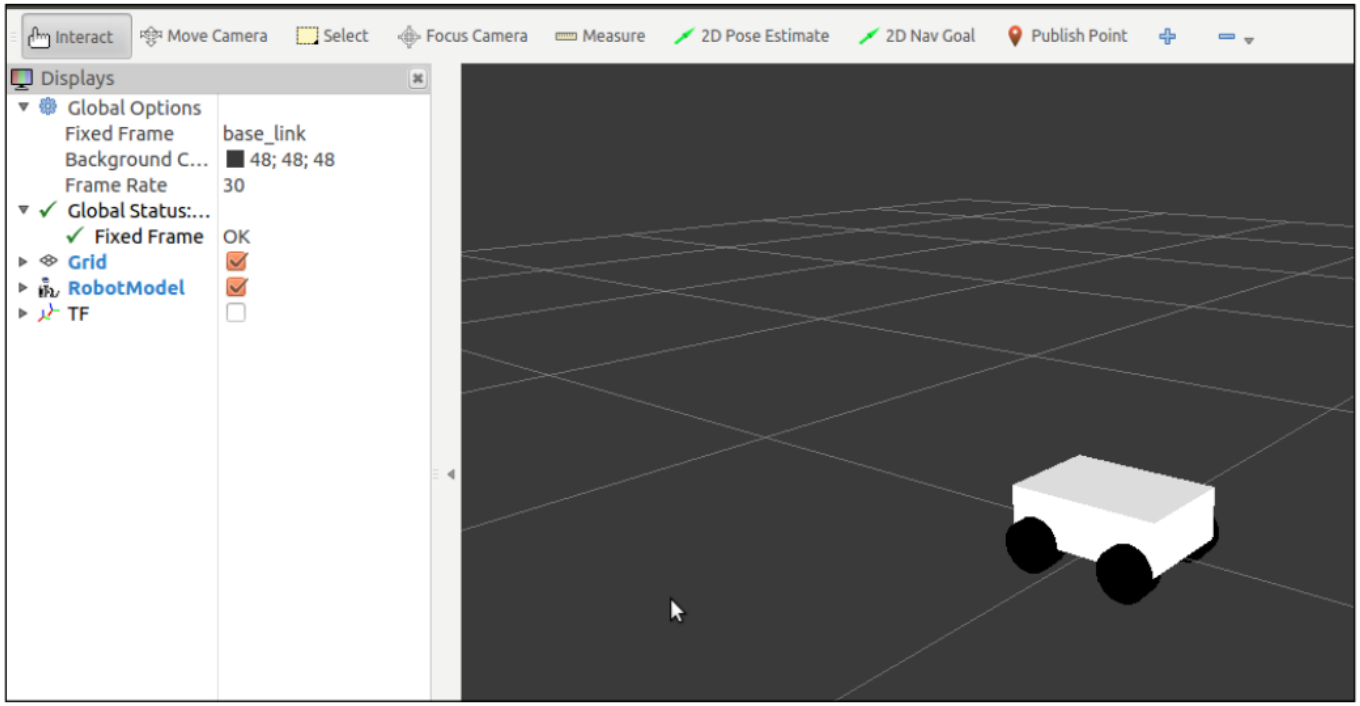
4. 在rviz里查看3D模型

在robot1_description/launch文件夹下创建display.launch文件，
并在文件中输入以下代码：

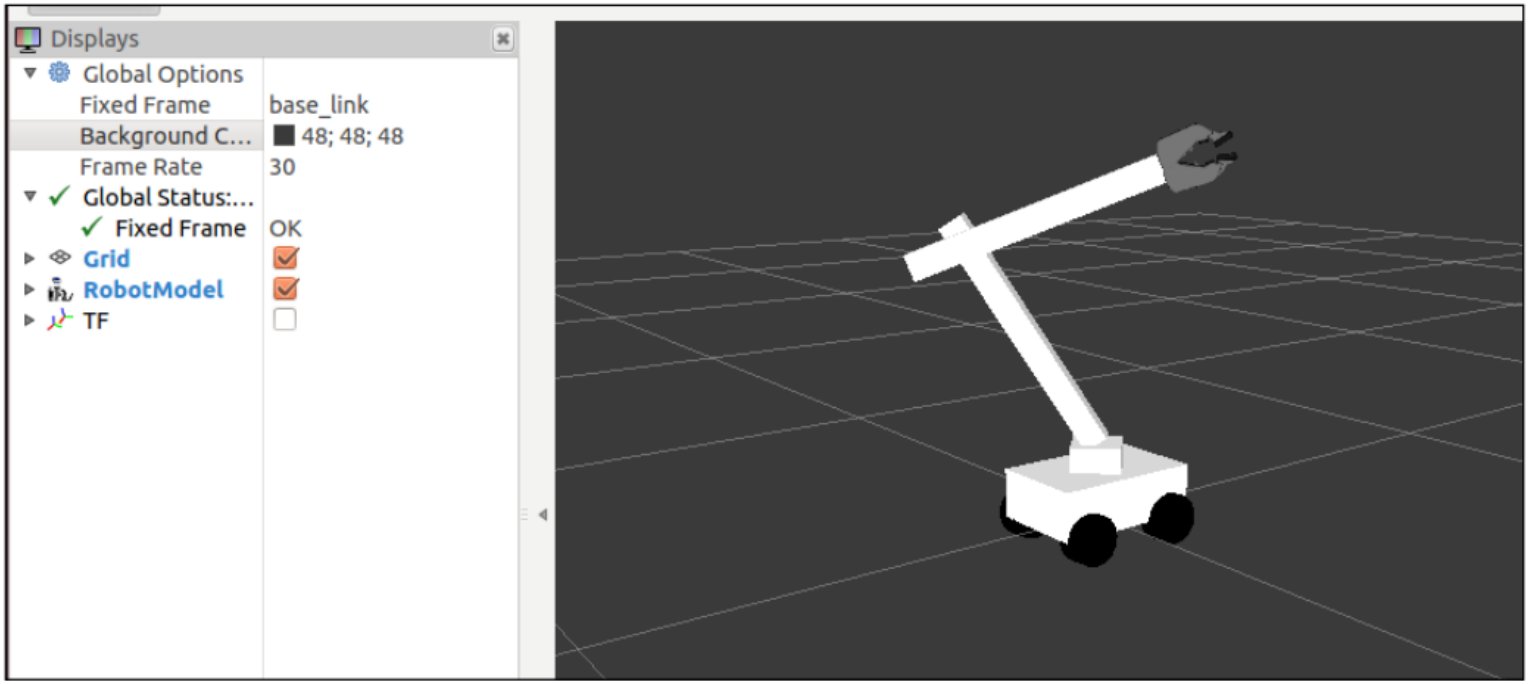
```
<?xml version="1.0"?>

<launch>
  <arg name="model" />
  <arg name="gui" default="False" />
  <param name="robot_description" textfile="$(arg model)" />
  <param name="use_gui" value="$(arg gui)" />
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />
</launch>
```

使用以下命令启动它：

```
$ roslaunch robot1_description display.launch
model:="`rospack find
robot1_description`/urdf/robot1.urdf"
```



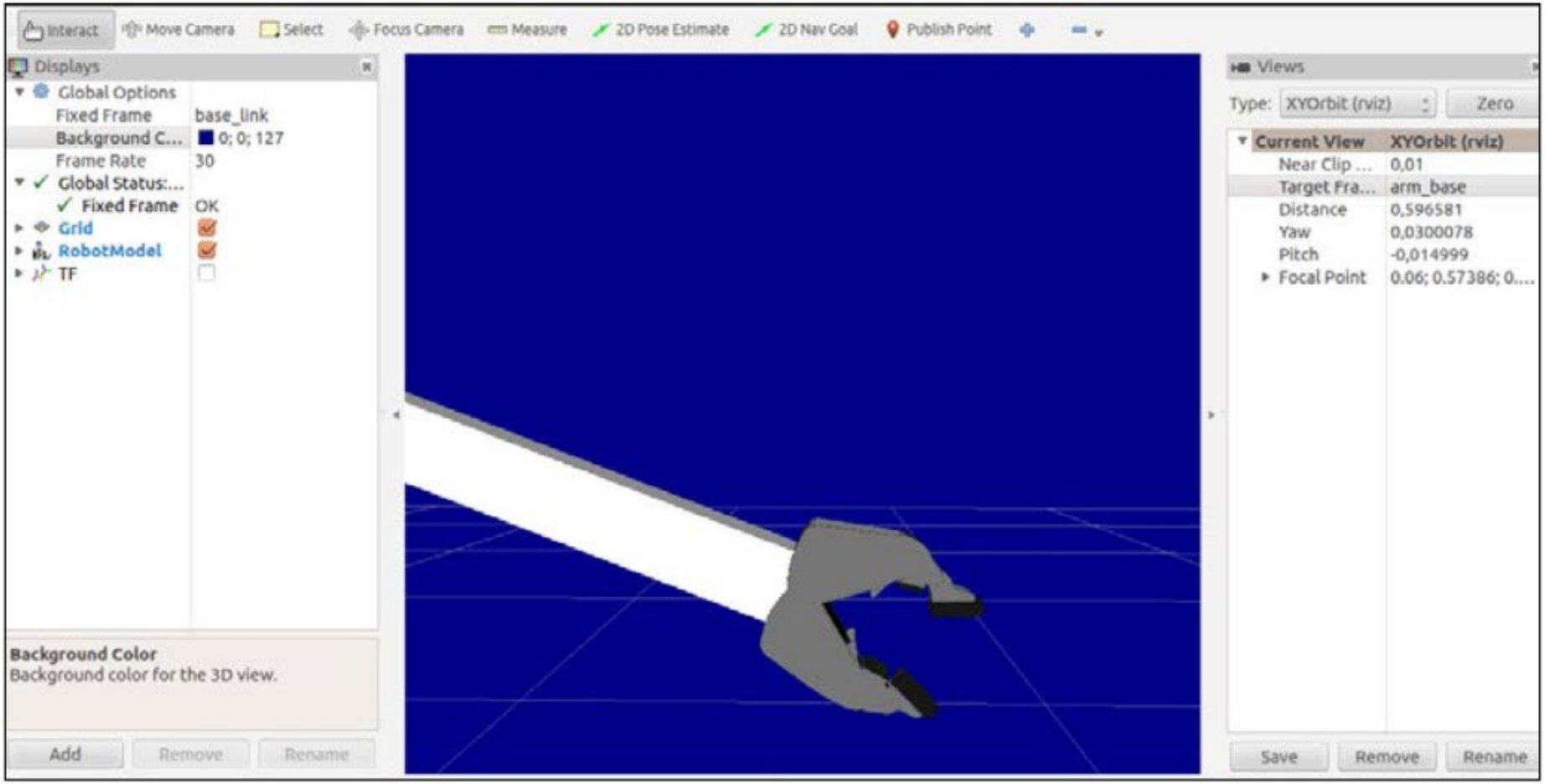
添加一些组件来完成设计：一段基座臂、一段连接臂和一个夹持器。可以在chapter7_tutorials/robot1_description/urdf/robot1.urdf文件中找到最终的模型。



5. 加载网格到机器人模型

有时候你希望自己构建的模型能够更加真实，并不是说简单地增加更多的基本几何形状和模块，而是通过添加更多的现实元素使模型变得更加丰富和细致。这就需要加载我们自行创建的网格（mesh）或者使用其他机器人模型的网格。

```
<link name="left_gripper">
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0" />
    <geometry>
      <mesh filename="package://pr2_description/meshes/gripper_v0/l_
finger.dae" />
    </geometry>
  </visual>
</link>
```


Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
base_link
Background C...
0; 0; 127
Frame Rate
30
Global Status:...
Fixed Frame
OK
Grid
RobotModel
TF
Background Color
Background color for the 3D view.
Add
Remove
Rename
Views
Type:
XYOrbit (rviz)
Zero
Current View
XYOrbit (rviz)
Near Clip ...
0,01
Target Fra...
arm_base
Distance
0,596581
Yaw
0,0300078
Pitch
-0,014999
Focal Point
0.06; 0.57386; 0....
Save
Remove
Rename

加载结果图

6. 使机器人模型运动

我们在arm_1_to_arm_base上使用的是转动关节，
其代码如下所示：

```
<joint name="arm_1_to_arm_base" type="revolute">
  <parent link="arm_base"/>
  <child link="arm_1"/>
  <axis xyz="1 0 0"/>
  <origin xyz="0 0 0.15"/>
  <limit effort ="1000.0" lower="-1.0" upper="1.0" velocity="0.5"/>
</joint>
```

限制转动角度

要判断关节的轴或转动限值是否合适，有一种好的办法就是使用Join_State_Publisher GUI运行rviz：

```
$ roslaunch robot1_description display.launch model:="`rospack find robot1_description`/urdf/robot1.urdf" gui:=true
```

7. 物理属性和碰撞属性

向名为wheel_1的连接中添加这些新参数：

```
<link name="wheel_1">
  ...
  <collision>
    <geometry>
      <cylinder length="0.05" radius="0.05"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
  </inertial>
</link>
```

记住，要为所有连接添加collision和inertial元素，因为如果你不这样做的话，Gazebo将无法使用这些模型

7.3 xacro——一个更好的机器人建模方法

xacro可帮助我们压缩URDF文件的大小，并且增加文件的可读性和可维护性。它还允许我们创建模型并复用这些模型去创建相同的结构，如更多的手臂和腿。

1. 使用常量

我们使用xacro声明常量，因此能够避免在很多行重复使用同一个数值。不使用xacro的话，若需要改变一个值，那就要修改若干个地方，从而很难进行文件的维护。

例如，四个轮子使用相同的长度和半径。如果我们希望修改这个值，那么需要在每一行进行修改。而如果使用如下的定义，那么修改就会变得轻松：

```
<xacro:property name="length_wheel" value="0.05" />
<xacro:property name="radius_wheel" value="0.05" />
```

现在，只需要使用${name_of_variable}来引用刚才想要修改的这些值：

```
${name_of_variable}:
<cylinder length="${length_wheel}"
radius="${radius_wheel}"/>
```

2. 使用数学方法

可以在${}结构中使用基本的四则运算（+、-、*、/）、一元负号和圆括号来构建任意复杂的表达式。但是不包括求幂和求模运算：

```
<cylinder radius="${wheeldiam/2}" length=".1"/>
<origin xyz="${reflect*(width+.02)} 0 .25" />
```

通过使用数学方法，我们能够通过修改某个值来更改模型的大小。而在此之前，我们需要先做好参数设计。

3. 使用宏

宏是xacro功能包中最有用的组件。为了能够更进一步减小文件，我们将会使用以下宏来做inertial初始化：

```
<xacro:macro name="default_inertial" params="mass">
  <inertial>
    <mass value="${mass}" />
    <inertia ixx="1.0" ixy="0.0" ixz="0.0"
     iyy="1.0" iyz="0.0"
     izz="1.0" />
  </inertial>
</xacro:macro>
<xacro:default_inertial mass="100"/>
```

为了在rviz和Gazebo中使用xacro文件，需要将它转换成.urdf文件。可以在robot1_description/urdf文件夹下执行以下命令来完成转换

```
$ rosrun xacro xacro.py robot1.xacro > robot1_processed.urdf
```

也可以在任意地方执行以下命令，它会起到和前面命令相同的作用：

```
$ rosrun xacro xacro.py "`rospack find robot1_description`/urdf/robot1.
xacro" > "`rospack find robot1_description`/urdf/robot1_processed.urdf"
```

4. 使用代码移动机器人

创建一个简单的节点来移动机器人。ROS提供了用于控制机器人的优秀工具，如ros_control功能包。在robot1_description/src文件夹下以state_publisher.cpp为名称创建一个新文件，并复制以下代码

```
#include <string>
#include <ros/ros.h>
#include <sensor_msgs/JointState.h>
#include <tf/transform_broadcaster.h>
int main(int argc, char** argv) {
  ros::init(argc, argv, "state_publisher");
  ros::NodeHandle n;
  ros::Publisher joint_pub = n.advertise<sensor_
msgs::JointState>("joint_states", 1);
  tf::TransformBroadcaster broadcaster;
```

```
    ros::Rate loop_rate(30);
    const double degree = M_PI/180;
    // robot state
    double inc= 0.005, base_arm_inc= 0.005, arm1_armbase_inc= 0.005,
arm2_arm1_inc= 0.005, gripper_inc= 0.005, tip_inc= 0.005;
    double angle= 0 ,base_arm = 0, arm1_armbase = 0, arm2_arm1 = 0,
gripper = 0, tip = 0;
  // message declarations
    geometry_msgs::TransformStamped odom_trans;
    sensor_msgs::JointState joint_state;
    odom_trans.header.frame_id = "odom";
    odom_trans.child_frame_id = "base_link";
while (ros::ok()) {
     //update joint_state
     joint_state.header.stamp = ros::Time::now();
     joint_state.name.resize(7);
     joint_state.position.resize(7);
     joint_state.name[0] ="base_to_arm_base";
     joint_state.position[0] = base_arm;
```

```
joint_state.name[1] ="arm_1_to_arm_base";
      joint_state.position[1] = arm1_armbase;
      joint_state.name[2] ="arm_2_to_arm_1";
      joint_state.position[2] = arm2_arm1;
      joint_state.name[3] ="left_gripper_joint";
      joint_state.position[3] = gripper;
      joint_state.name[4] ="left_tip_joint";
      joint_state.position[4] = tip;
      joint_state.name[5] ="right_gripper_joint";
      joint_state.position[5] = gripper;
      joint_state.name[6] ="right_tip_joint";
      joint_state.position[6] = tip;
     // update transform
     // (moving in a circle with radius 1)
     odom_trans.header.stamp = ros::Time::now();
     odom_trans.transform.translation.x = cos(angle);
     odom_trans.transform.translation.y = sin(angle);
     odom_trans.transform.translation.z = 0.0;
```

创建launch文件来启动节点、模型和所有必要的组件。在robot1_description/launch文件夹下以display_xacro.launch为名创建一个新文件（内容如下）：

```
<?xml version="1.0"?>

<launch>
  <arg name="model" />
  <arg name="gui" default="False" />
  <param name="robot_description" command="$(find xacro)/xacro.py
$(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>
  <node name="state_publisher_tutorials" pkg="robot1_description"
type="state_publisher_tutorials" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find robot1_
description)/urdf.rviz" />
</launch>
```

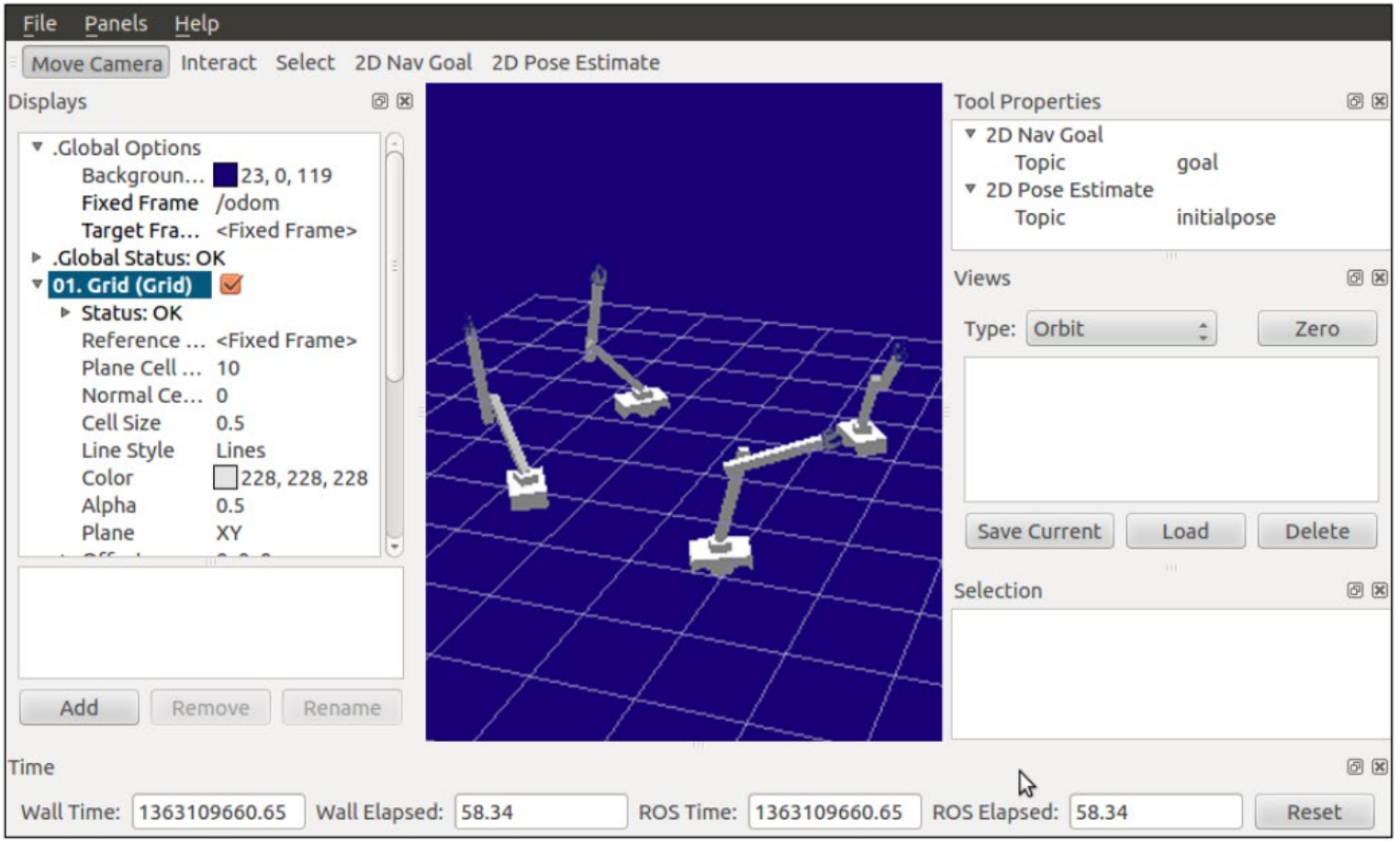
启动该节点前，必须安装以下功能包：

```
$ sudo apt-get instal ros-kinetic-map-server
$ sudo apt-get install ros-kinetic-fake-localization
$ cd ~/dev/catkin_ws && catkin_make
```

Hydro—12.04; indigo---14.04; kinetic---16.04; Melodic—18.04

使用以下命令，启动带有完整模型的新节点。我们将会在rviz中看到3D模型的每一个关节都在运动：

```
$ roslaunch robot1_description state_xacro.launch model:="`rospack find robot1_description`/urdf/robot1.xacro"
```

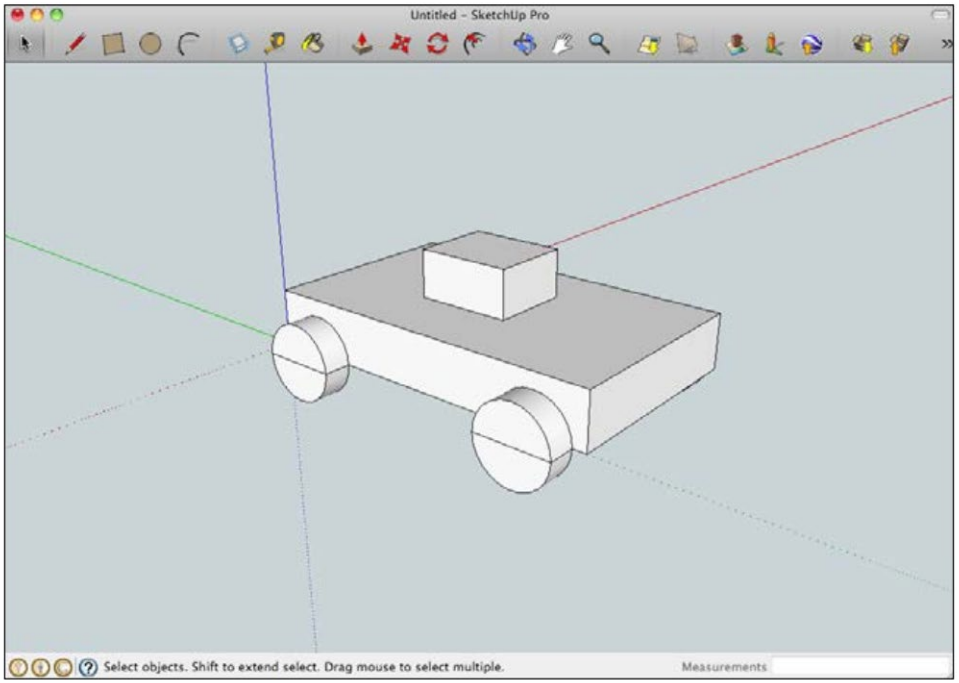

File Panels Help
Move Camera Interact Select 2D Nav Goal 2D Pose Estimate
Displays
.Global Options
Backgroun... 23, 0, 119
Fixed Frame /odom
Target Fra... <Fixed Frame>
.Global Status: OK
01. Grid (Grid)
Status: OK
Reference ... <Fixed Frame>
Plane Cell ... 10
Normal Ce... 0
Cell Size 0.5
Line Style Lines
Color 228, 228, 228
Alpha 0.5
Plane XY
Add Remove Rename
Tool Properties
2D Nav Goal
Topic goal
2D Pose Estimate
Topic initialpose
Views
Type: Orbit Zero
Save Current Load Delete
Selection
Time
Wall Time: 1363109660.65 Wall Elapsed: 58.34 ROS Time: 1363109660.65 ROS Elapsed: 58.34 Reset

rviz中的3D模型

5. 使用SketchUp进行3D建模

请注意，SketchUp只能在Windows系统或Mac系统下运行，而本节的模型是在Mac中开发的，而不是在Linux中。

首先，你需要在电脑中安装SketchUp。安装好之后，创建一个和下图类似的模型：



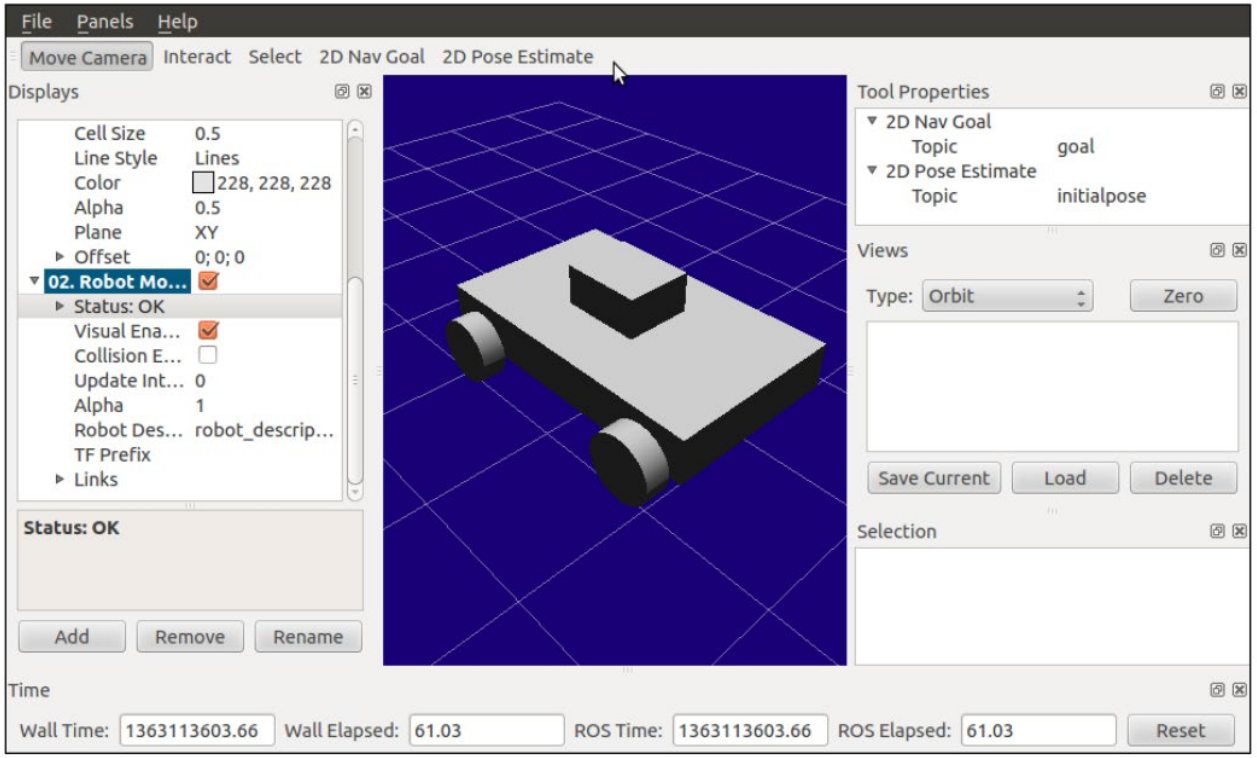
为了导出模型，从菜单栏上找到Export|3D Model|Save As COLLADA File（*.dae）。我们将文件命名为bot.dae，并将这个文件保存在robot1_description/meshes文件夹下

现在，为了能使用3D模型，我们将在robot1_description/urdf文件夹下创建一个以dae.urdf为名的新文件。在文件中添加以下代码：

```
<?xml version="1.0"?>
<robot name="robot1">
   <link name="base_link">
   <visual>
      <geometry>
         <mesh scale="0.025 0.025 0.025“ filename="package://robot1_
description/meshes/bot.dae"/>
      </geometry>
      <origin xyz="0 0 0.226"/>
   </visual>
   </link>
</robot>
```

使用以下命令测试模型：

$ roslaunch robot1_description display.launch model:="`rospack find robot1_description`/urdf/dae.urdf"



7.4 在ROS中仿真

- Gazebo（http://gazebosim.org/ ）是一种适用于复杂室内多机器人和室外环境的仿真环境。它能够在三维环境中对多个机器人、传感器及物体进行仿真，产生实际传感器反馈和物体之间的物理响应。

- Gazebo现在独立于ROS，并在Ubuntu中以独立功能包安装。

1. 在Gazebo中使用URDF3D模型

安装Gazebo：

```
$ gazebo
```

安装ROS功能包与Gazebo交互：

```
$ sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control
```

使用以下命令测试Gazebo与ROS的集成，并检查GUI是否打开：

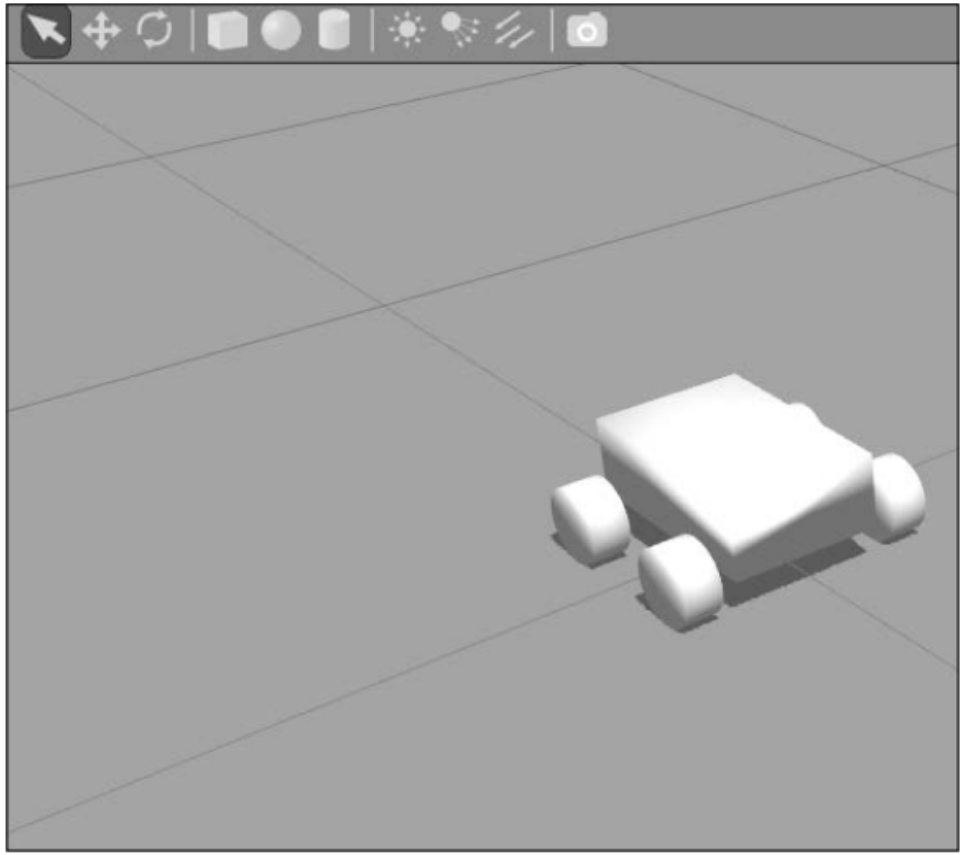
```
$ roscore & rosrun gazebo_ros gazebo
```

创建一个名为gazebo.launch的新文件到chapter7_tutorials/robot1_gazebo/launch文件夹下，并添加以下代码：

```
<?xml version="1.0"?>

<launch>
  <!-- these are the arguments you can pass this launch file, for
example paused:=true -->
  <arg name="paused" default="true" />
  <arg name="use_sim_time" default="false" />
  <arg name="gui" default="true" />
  <arg name="headless" default="false" />
  <arg name="debug" default="true" />
  <!-- We resume the logic in empty_world.launch, changing only the
name of the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find robot1_gazebo)/worlds/robot.
world" />
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)" />
    <arg name="use_sim_time" value="$(arg use_sim_time)" />
    <arg name="headless" value="$(arg headless)" />
  </include>
  <!-- Load the URDF into the ROS Parameter Server -->
  <arg name="model" />
  <param name="robot_description" command="$(find xacro)/xacro.py
$(arg model)" />
  <!-- Run a python script to the send a service call to gazebo_ros to
spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen" args="-urdf -model robot1 -param
robot_description -z 0.05" />
</launch>
```

启动文件，需要使用以下命令：

```
$ roslaunch robot1_gazebo gazebo.launch model:="`rospack find
robot1_description`/urdf/robot1_base_01.xacro"
```



为了在Gazebo中添加可见的纹理，需要在你的.gazebo模型文件中使用以下代码在robot1_description/urdf中创建robot.gazebo：

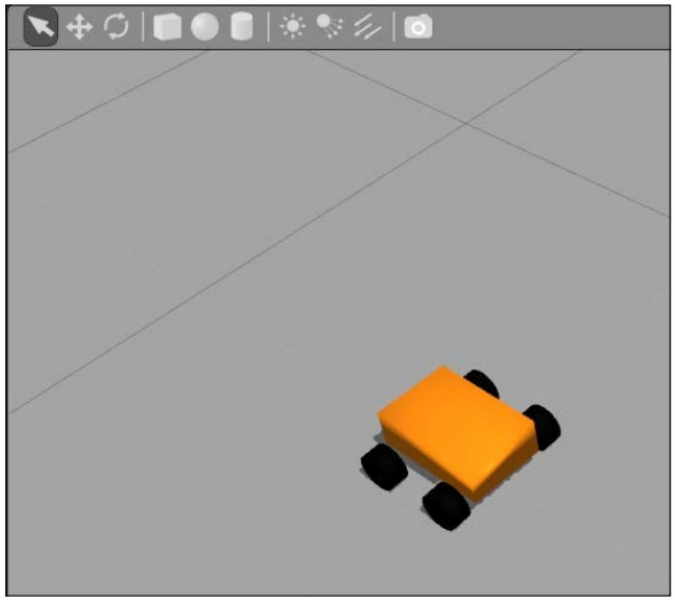
```
<gazebo reference="base_link">
  <material>Gazebo/Orange</material>
</gazebo>
<gazebo reference="wheel_1">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_2">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_3">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_4">
  <material>Gazebo/Black</material>
</gazebo>
```

将robot1_description/urdf/robot1_base_01.xacro文件另存为robot1_base_02.xacro，并添加以下代码：

```
<xacro:include filename="$(find robot1_description)/urdf/robot.gazebo" />
```

启动这个新文件：

```
$ roslaunch robot1_gazebo gazebo.launch model:="`rospack find robot1_description`/urdf/robot1_base_02.xacro"
```



2. 在Gazebo中添加传感器

向.xacro文件中增加这些行来为机器人添加Hokuyo激光雷达3D模型：

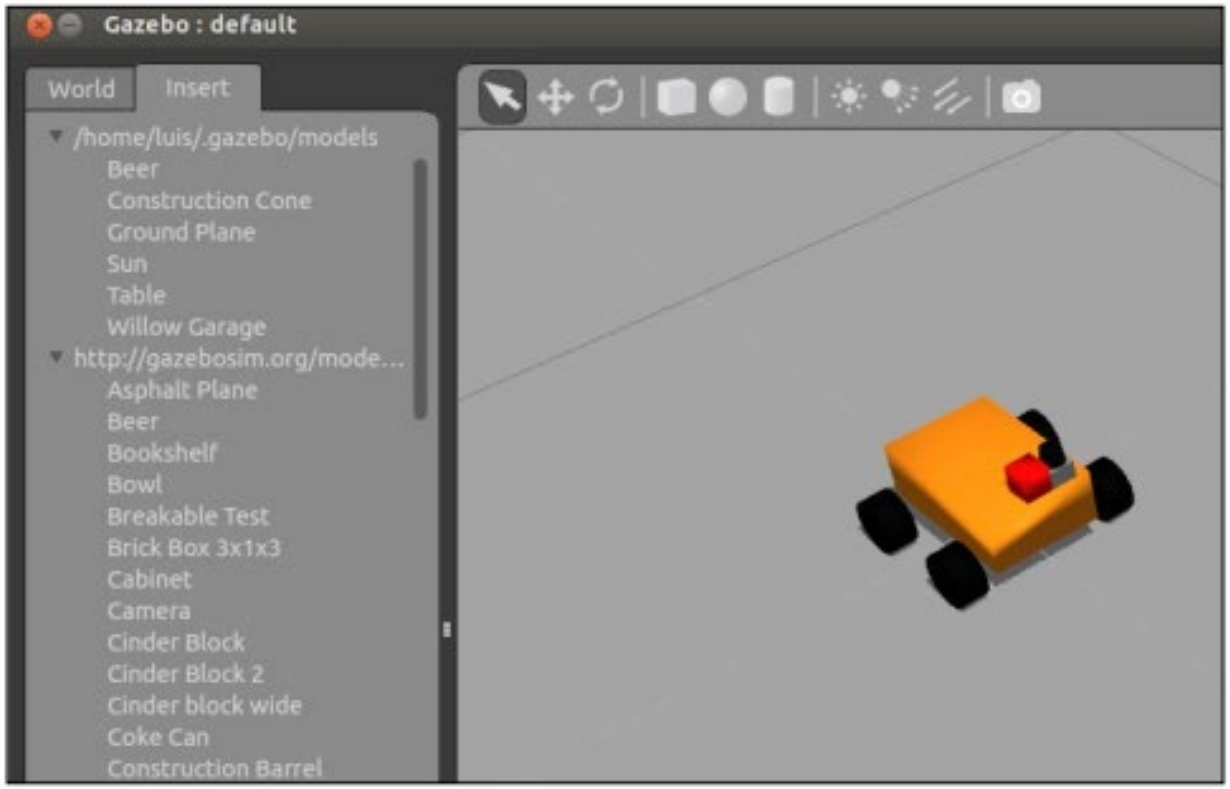
```
<?xml version="1.0" encoding="UTF-8"?>
<link name="hokuyo_link">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="0.1 0.1 0.1" />
    </geometry>
  </collision>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://robot1_description/meshes/hokuyo.dae"
/>
    </geometry>
  </visual>
  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0" />
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"
/>
  </inertial>
</link>
```

在.gazebo文件里，我们将添加libgazebo_ros_laser插件，这样就可以模拟Hokuyo激光测距雷达的行为：

```
<gazebo reference="hokuyo_link">
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>false</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser

             achieving "+-30mm" accuracy at range < 10m.  A mean of
0.0m and

             stddev of 0.01m will put 99.7% of samples within 0.03m
of the true

             reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
      <topicName>/robot/laser/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

使用以下命令启动新的模型：

$ roslaunch robot1_gazebo gazebo.launch model:="`rospack find robot1_description`/urdf/robot1_base_03.xacro"

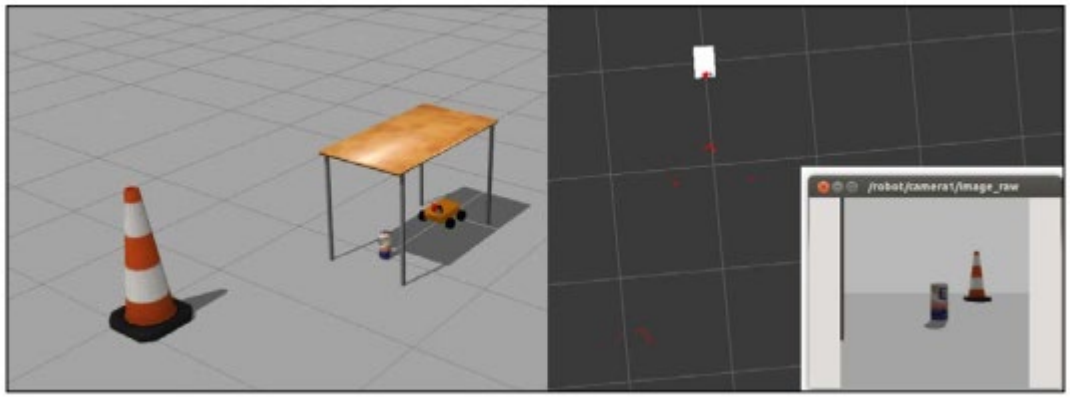


通过rostopic echo命令看到雷达产生的传感器数据：

$ rostopic echo /robot/laser/scan

观察摄像头看到的gazebo仿真图像，在终端中写入以下指令：

$ rosrun image_view image_view image:=/robot/camera1/image_raw

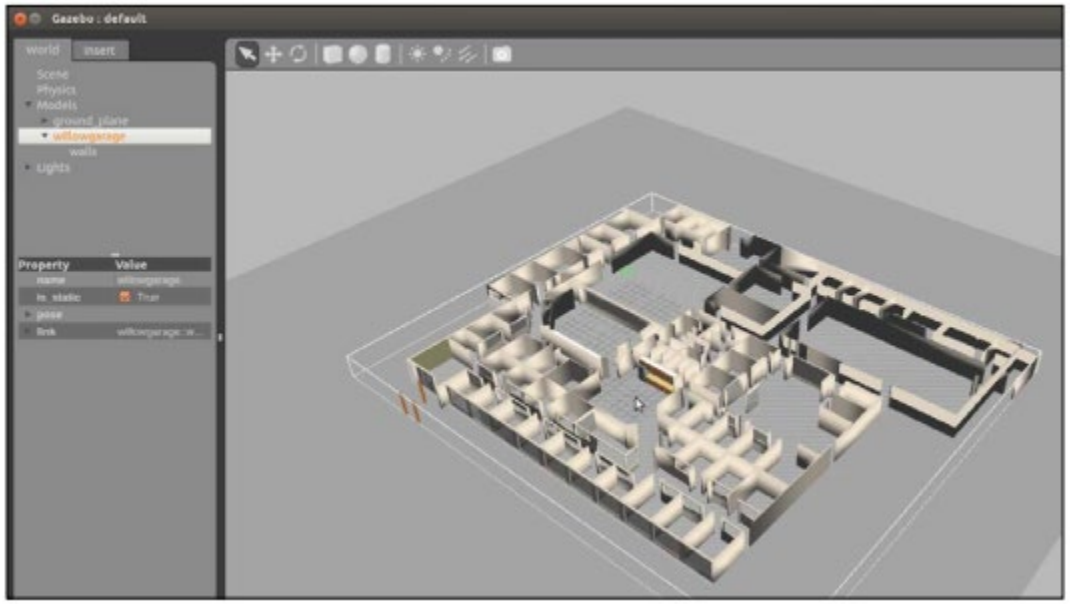


3. 在Gazebo中加载和使用地图

在本小节中我们将会使用一张柳树车库公司（Willow Garage）办公室的地图，这张地图在我们的ROS软件中应该已经默认安装了。

使用以下命令启动.launch文件：

```
$ roslaunch gazebo_ros willowgarage_world.launch
```

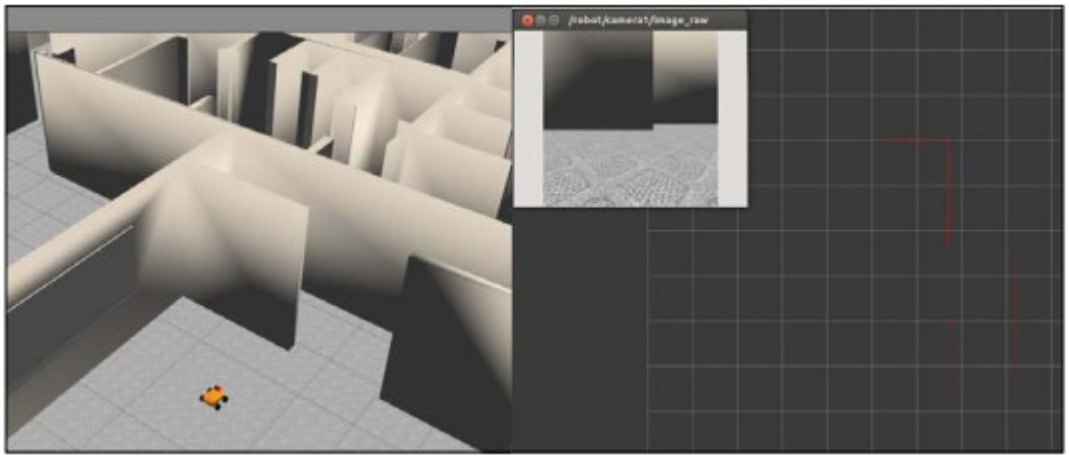


创建一个新的.launch文件来同时加载地图和机器人。为实现这个功能，我们在robot1_gazebo/launch文件夹下创建一个名为gazebo_wg.launch的文件，并添加以下代码：

```
<?xml version="1.0"?>
<launch>
  <include file="$(find gazebo_ros)/launch/willowgarage_world.launch">
  </include>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description"
command="$(find xacro)/xacro.py '$(find robot1_description)/
urdf/robot1_base_03.xacro'" />

  <!-- Run a python script to the send a service call to gazebo_ros to
spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen" args="-urdf -model robot1 -param
robot_description -z 0.05"/>
</launch>
```

现在运行带有激光雷达的模型文件：

```
$ roslaunch robot1_gazebo gazebo_wg.launch
```



4. 在Gazebo中移动机器人

和前面的激光雷达一样，Gazebo也已经有了skid驱动的实现，我们能够直接使用它移动机器人。使用此控制器，只需要在模型文件中增加以下代码：

```
<gazebo>
  <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_
skid_steer_drive.so">
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>base_to_wheel1</leftFrontJoint>
    <rightFrontJoint>base_to_wheel3</rightFrontJoint>
    <leftRearJoint>base_to_wheel2</leftRearJoint>
    <rightRearJoint>base_to_wheel4</rightRearJoint>
    <wheelSeparation>4</wheelSeparation>
    <wheelDiameter>0.1</wheelDiameter>
    <robotBaseFrame>base_link</robotBaseFrame>
    <torque>1</torque>
    <topicName>cmd_vel</topicName>
    <broadcastTF>0</broadcastTF>
</plugin>
</gazebo>
```

使用以下命令启动带有控制器和地图的模型：

```
$ roslaunch robot1_gazebo gazebo_wg.launch
```

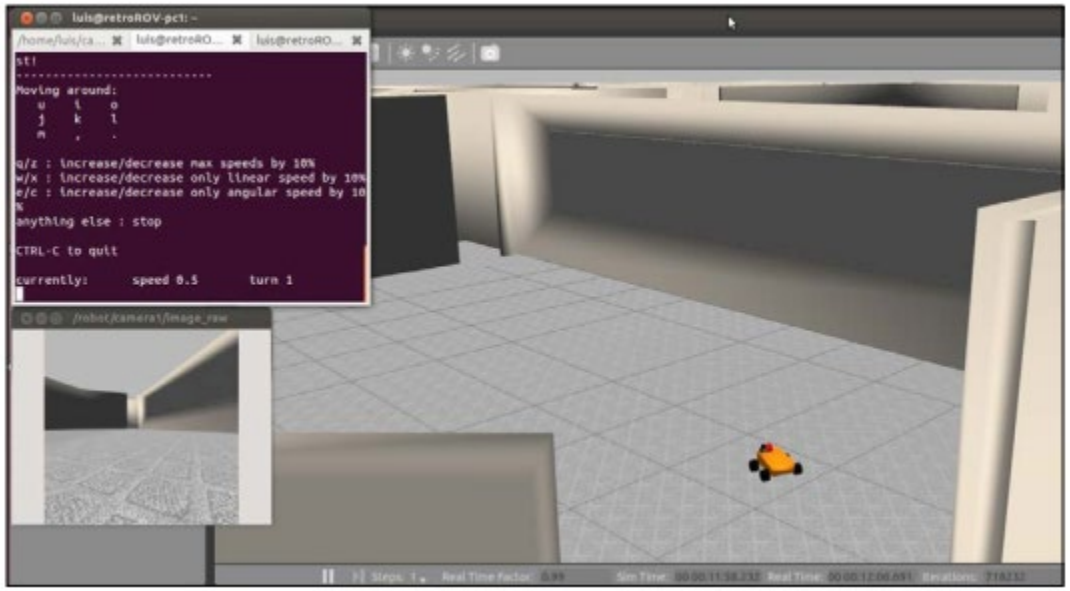
我们将使用键盘来移动地图中的机器人。这个节点在teleop_twist_keyboard功能包中，发布/cmd_vel主题。运行以下命令以安装此功能包：

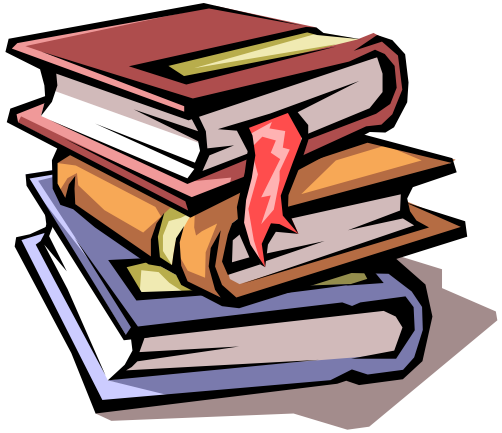
```
$ sudo apt-get install ros-kinetic-teleop-twist-keyboard
$ rosstack profile
$ rospack profile
```

然后，运行节点如下：

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

现在你会看到一个有很多说明的新命令行窗口，你可以使用（u，i，o，j，k，l，m，"，"，"."）按键来移动机器人并设置最大速度。





Q & A

Thanks !