ROS可视化——gazebo

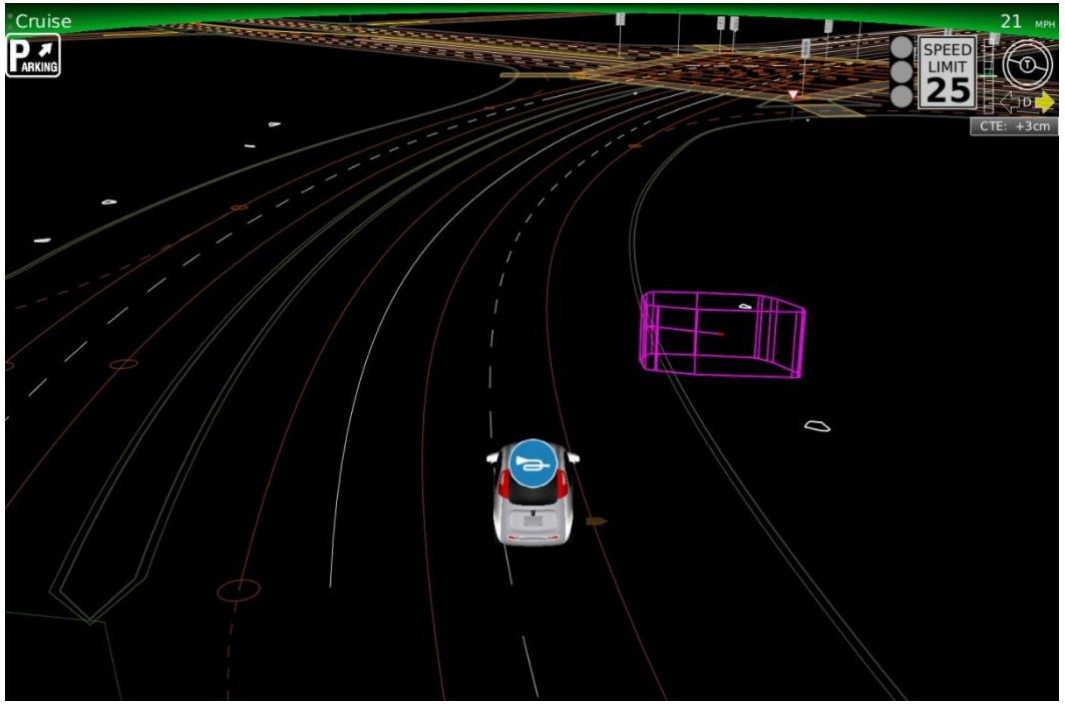
主讲人：胡春旭
时　间：2017年7月24日
邮　箱：huchunxu@aigalaxy.com
古月居：http://www.guyuehome.com

星河智能
AIgalaxy

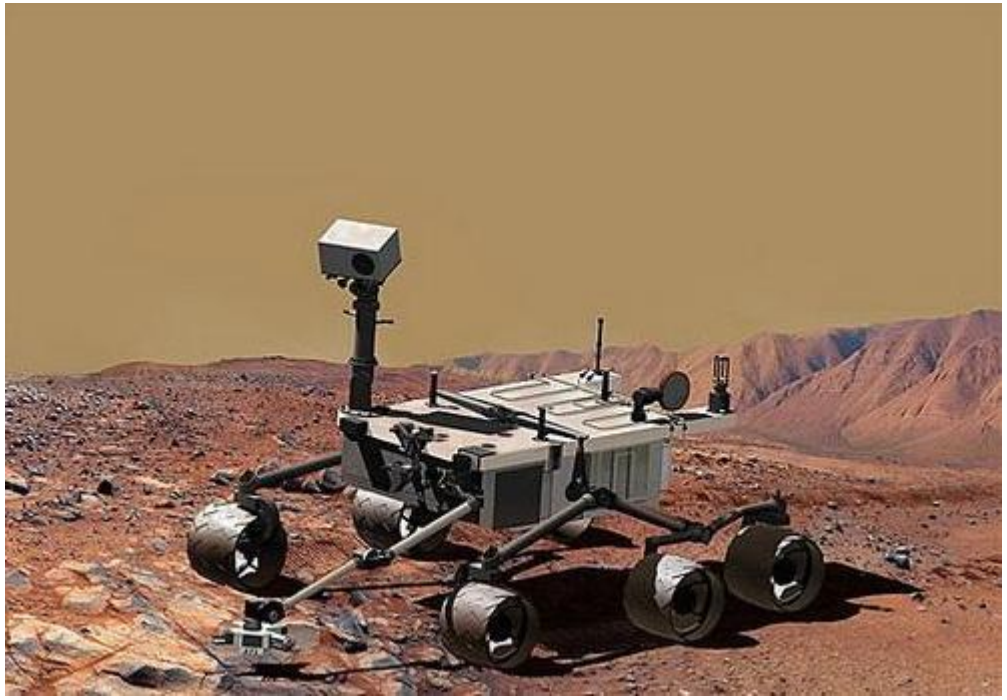
目录 Contents

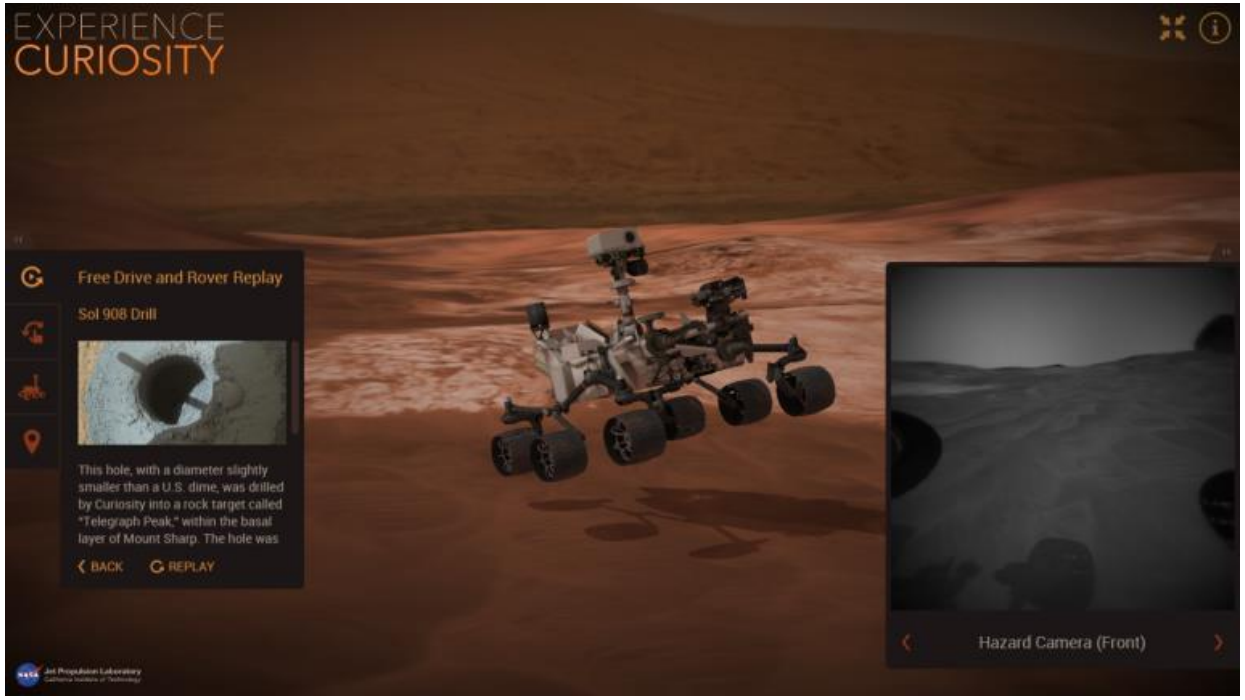
- Why-为什么要用gazebo
- What-什么是gazebo
- How-如何使用gazebo

Why-无人驾驶汽车的示例



Why-好奇号火星探测器的示例





Why-仿真的定义

- **定义** - 仿真/模拟（Simulation），泛指基于实验或训练的目的，将原本的系统、事务或流程，建立一个模型以表征其关键特性(key characteristics)或者行为/功能，予以系统化与公式化，以便进行可对关键特征做出模拟。（维基百科）

- **应用场景** - 当所研究的系统造价昂贵、实验的危险性大或需要很长的时间才能了解系统参数变化所引起的后果时，仿真是一种特别有效的研究手段。

- **应用领域** - 电气、机械、化工、水力、热力、社会、经济、生态、管理等等。

- **重要工具** - 计算机

- **ROS中的仿真软件** - gazebo、stage、matlab等

Why-Why gazebo?

- gazebo是一款功能强大的**三维物理仿真平台**
  - 具备强大的物理引擎
  - 高质量的图形渲染
  - 方便的编程与图形接口
  - 开源免费

- 其典型**应用场景**包括
  - 测试机器人算法
  - 机器人的设计
  - 现实情景下的回溯测试

Why-Robot simulation made easy

Features

Dynamics Simulation

Access multiple high-performance physics engines including ODE, Bullet, Simbody, and DART.

Advanced 3D Graphics

Utilizing OGRE, Gazebo provides realistic rendering of environments including high-quality lighting, shadows, and textures.

Sensors and Noise

Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and more.

Plugins

Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's API.

Robot Models

Many robots are provided including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. Or build your own using SDF.

TCP/IP Transport

Run simulation on remote servers, and interface to Gazebo through socket-based message passing using Google Protobufs.

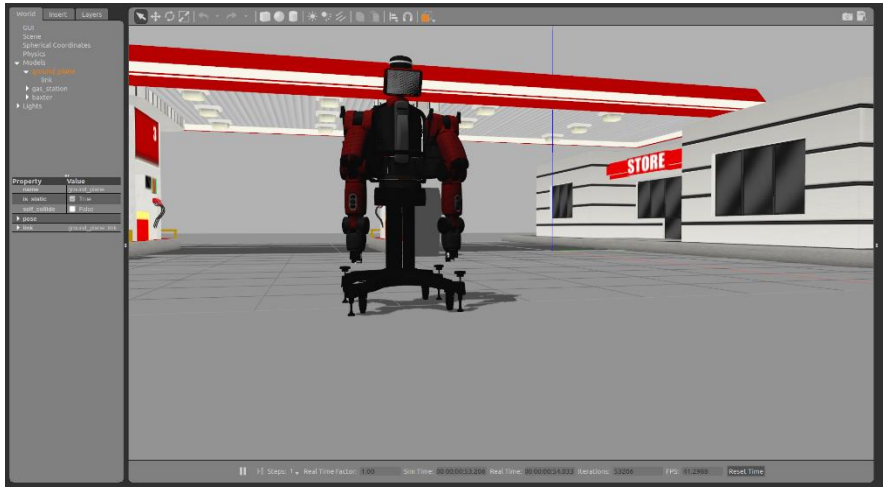
Cloud Simulation

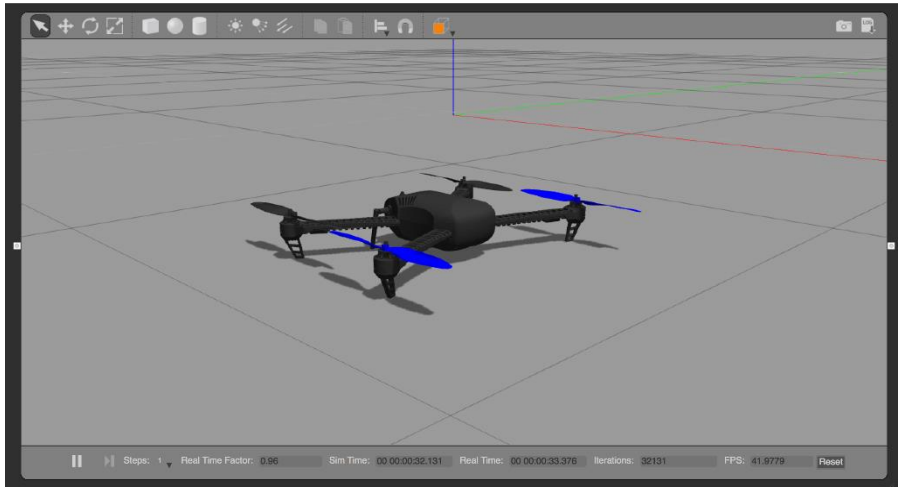
Use CloudSim to run Gazebo on Amazon, Softlayer, or your own OpenStack instance.

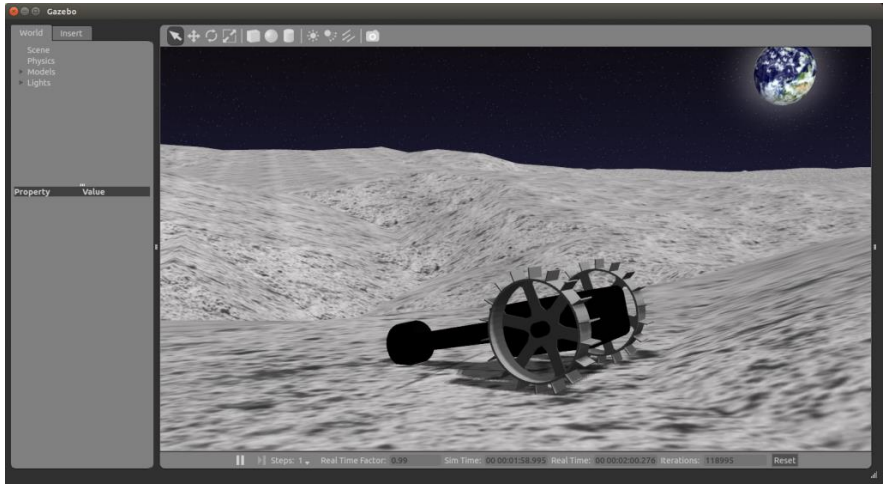
Command Line Tools

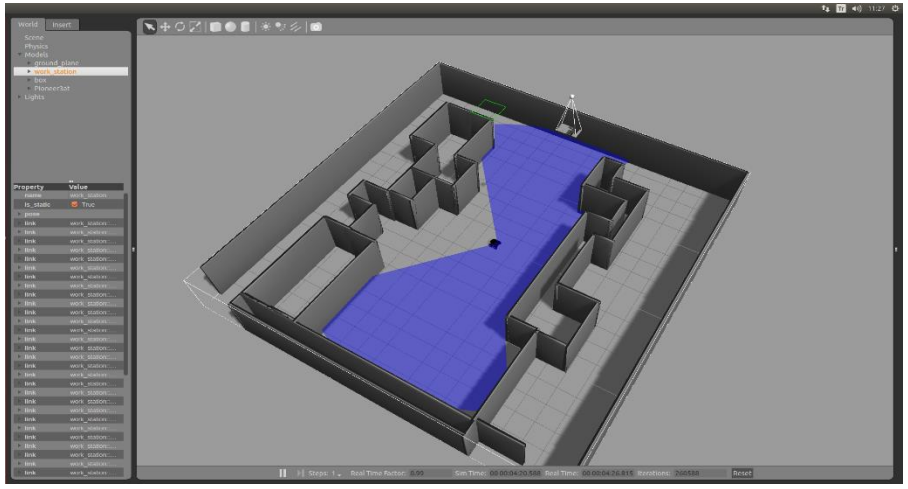
Extensive command line tools facilitate simulation introspection and control.

Why-Robot simulation made easy









What-gazebo overview

gazebo overview：https://www.youtube.com/watch?v=SK5GuC1jgg8

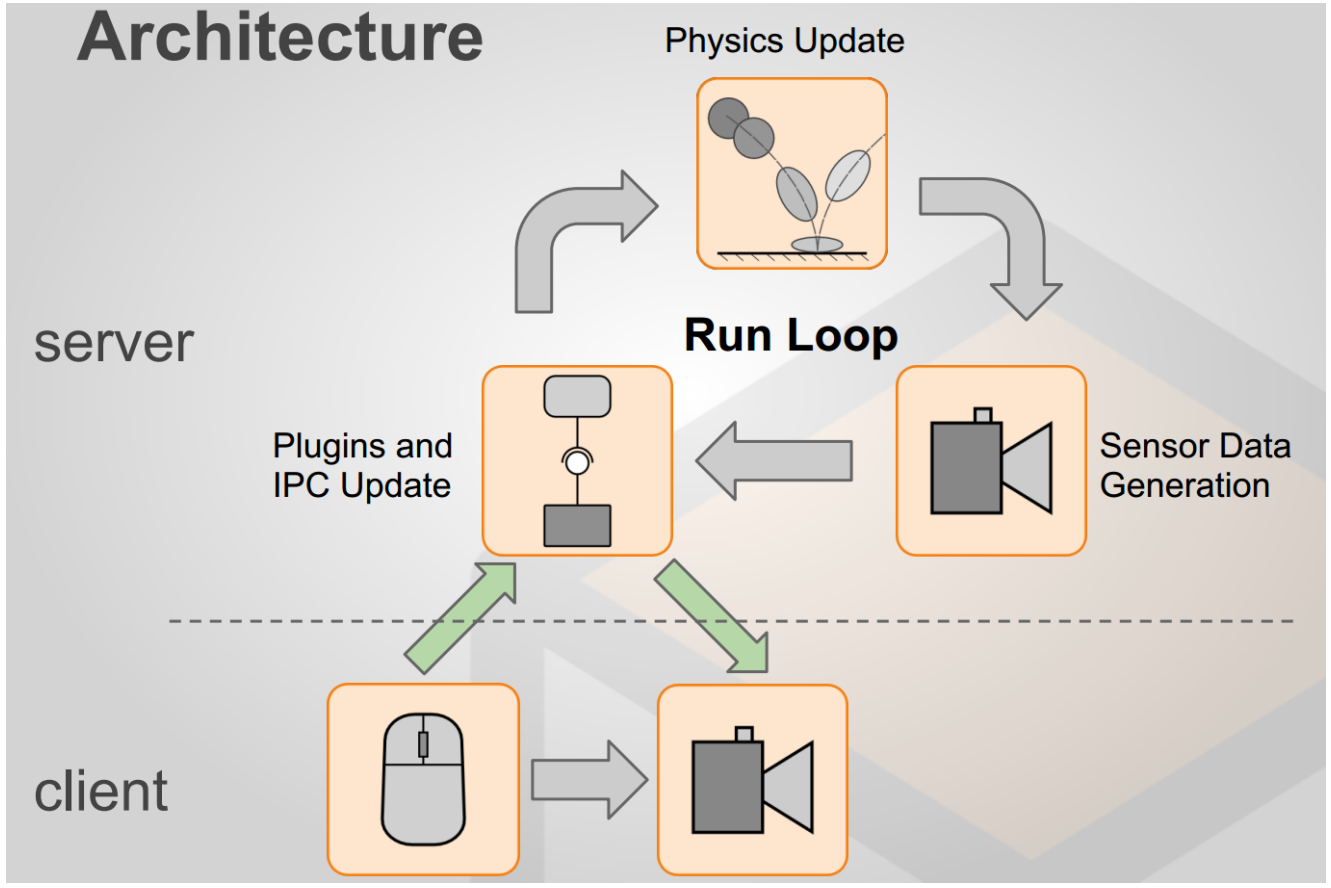
What-Architecture

➢ gzserver

- 产生传感器数据
- 更新物理参数
- 可以运行在云端

➢ gzclient

- 基于QT的用户界面
- 控制仿真特性
- 可同时运行多个人机界面



What-版本进化

Ver2.2

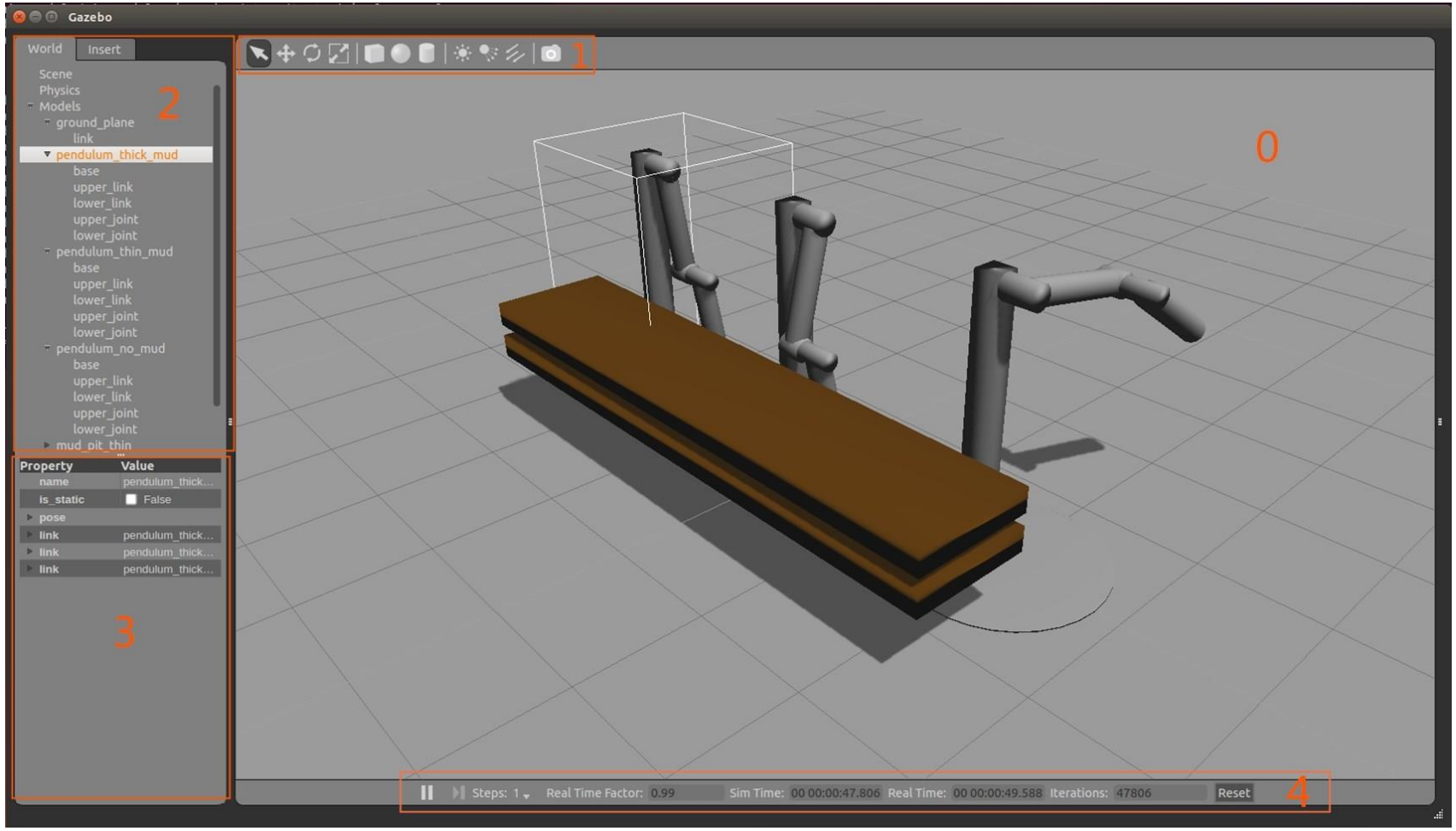
Ver5.0

Ver7.0

Gazebo 1.9	2013-07-24	EOL 2015-07-27
Gazebo 2.2	2013-11-07	Ubuntu P,Q,R,S EOL 2016-01-25
Gazebo 3.0	2014-04-11	Ubuntu P,R,S,T EOL 2015-07-27
Gazebo 4.0	2014-07-28	Ubuntu P,S,T EOL 2016-01-25
Gazebo 5.0	2015-01-26	Ubuntu T,U,V EOL 2017-01-25
Gazebo 6.0	2015-07-27	Ubuntu T,U,V EOL 2017-01-25
Gazebo 7.1	2016-01-25	Ubuntu T,V,W EOL 2021-01-25
Gazebo 8.0	2017-01-25	Ubuntu X,Y EOL 2019-01-25
Gazebo 9.0	2018-01-25	EOL 2023-01-25
Gazebo 10.0	2019-01-24	EOL 2021-01-24
Gazebo 11.0	2020-01-29	EOL 2025-01-29

What-界面布局

- 0 ：3D视图区
- 1 ：工具栏
- 2 ：模型列表
- 3 ：模型属性项
- 4 ：时间显示区



What-与ROS的接口

- **gazebo_ros** - 主要用于gazebo接口封装、gazebo服务端和客户端的启动、URDF模型生成等。
- **gazebo_msgs** - 是 gazebo 的Msg和Srv数据结构。
- **gazebo_plugins** - 用于gazebo的通用传感器插件。
- **gazebo_ros_api_plugin** 和 **gazebo_ros_path_plugin**这两个Gazebo的插件实现接口封装。

How-安装

安装

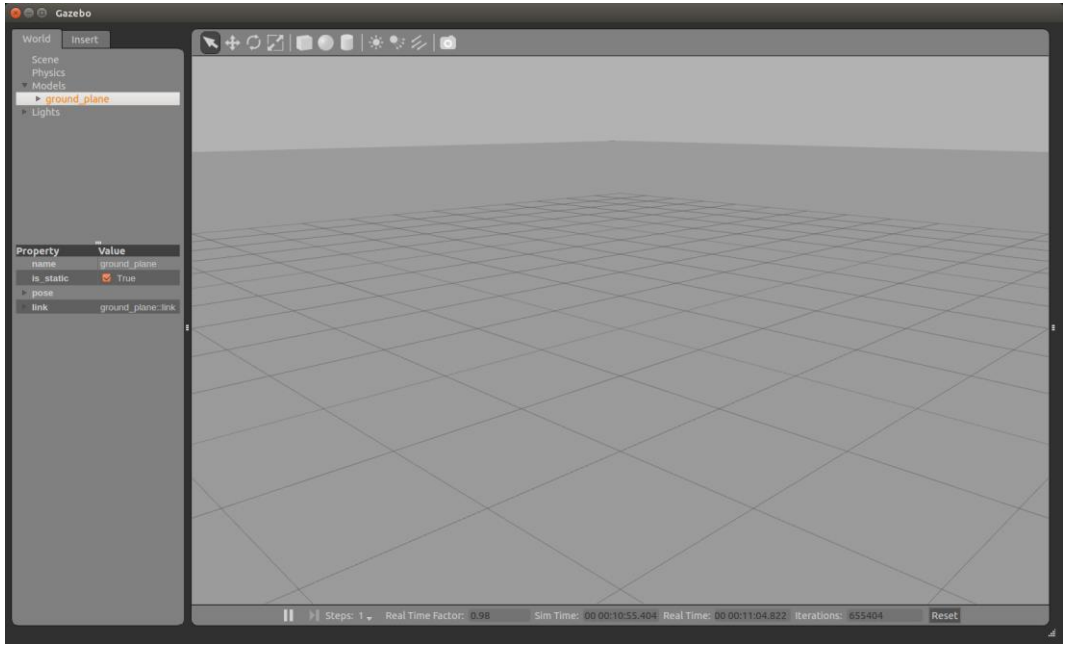
如果你已经成功安装了桌面完整版的ROS，可以直接跳过这一步骤，否则，请使用如下命令进行安装：

sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control

命令启动

roscore

rosrun gazebo_ros gazebo



launch启动

roslaunch gazebo_ros empty_world.launch

How-机器人仿真的步骤

创建仿真环境

修改机器人模型

开始仿真

How-empty_world.launch

```
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="extra_gazebo_args" default=""/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>
  <arg name="verbose" default="false"/>
  <arg name="world_name" default="worlds/empty.world"/> <!-- Note: the world_name is with respect to GAZEBO_RESOURCE_PATH environmental variable -->

  <!-- set use_sim_time flag -->
  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <!-- set command arguments -->
  <arg unless="$(arg paused)" name="command_arg1" value=""/>
  <arg     if="$(arg paused)" name="command_arg1" value="-u"/>
  <arg unless="$(arg headless)" name="command_arg2" value=""/>
  <arg     if="$(arg headless)" name="command_arg2" value="-r"/>
  <arg unless="$(arg verbose)" name="command_arg3" value=""/>
  <arg     if="$(arg verbose)" name="command_arg3" value="--verbose"/>
  <arg unless="$(arg debug)" name="script_type" value="gzserver"/>
  <arg     if="$(arg debug)" name="script_type" value="debug"/>

  <!-- start gazebo server-->
  <node name="gazebo" pkg="gazebo_ros" type="$(arg script_type)" respawn="false" output="screen"
    args="$(arg command_arg1) $(arg command_arg2) $(arg command_arg3) -e $(arg physics) $(arg extra_gazebo_args) $(arg world_name)" />

  <!-- start gazebo client -->
  <group if="$(arg gui)">
    <node name="gazebo_gui" pkg="gazebo_ros" type="gzclient" respawn="false" output="screen"/>
  </group>

</launch>
```

How-参数

```
<!-- these are the arguments you can pass this launch file, for example paused:=true -->
<arg name="paused" default="false"/>
<arg name="use_sim_time" default="true"/>
<arg name="extra_gazebo_args" default=""/>
<arg name="gui" default="true"/>
<arg name="headless" default="false"/>
<arg name="debug" default="false"/>
<arg name="physics" default="ode"/>
<arg name="verbose" default="false"/>
```

- **paused** - 以暂停状态启动*Gazebo*，默认为*false*。
- **use_sim_time** - ROS的*node*是否使用*Gazebo*通过*/clock topic*发布的仿真时间，默认为*true*。
- **gui** - 启用*Gazebo*的用户交互接口来控制视图，默认为*true*。
- **headless** - 禁用仿真渲染组件的功能，当*gui:=true*时，不能使能此参数，默认为*false*。
- **debug** - 在*gdb*中启动*gzserver(Gazebo Server)*以用于调试，默认为*false*。

How-world model

```
<arg name="world_name" default="worlds/empty.world"/> <!-- Note: the world_name is with respect to GAZEBO_RESOURCE_PATH environmental variable -->
```

- world_name以**相对路径**指定了world模型的定义文件，**绝对路径**由GAZEBO_RESOURCE_PATH这个环境变量决定。默认为/usr/share/gazebo-x.x，对indigo版本来说为/usr/share/gazebo-2.2

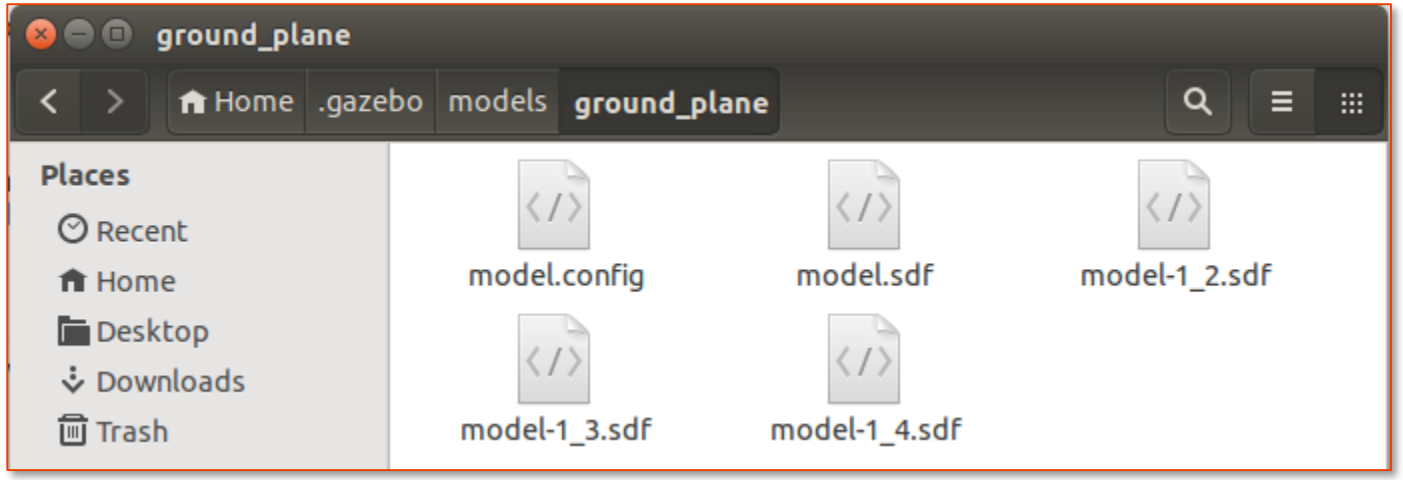
- **默认本地模型数据库路径**为~/.gazebo/models

empty.world

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>
  </world>
</sdf>
```

建议：为保证模型顺利加载，请提前将模型文件下载并放置到本地路径下。
https://bitbucket.org/osrf/gazebo_models/downloads/

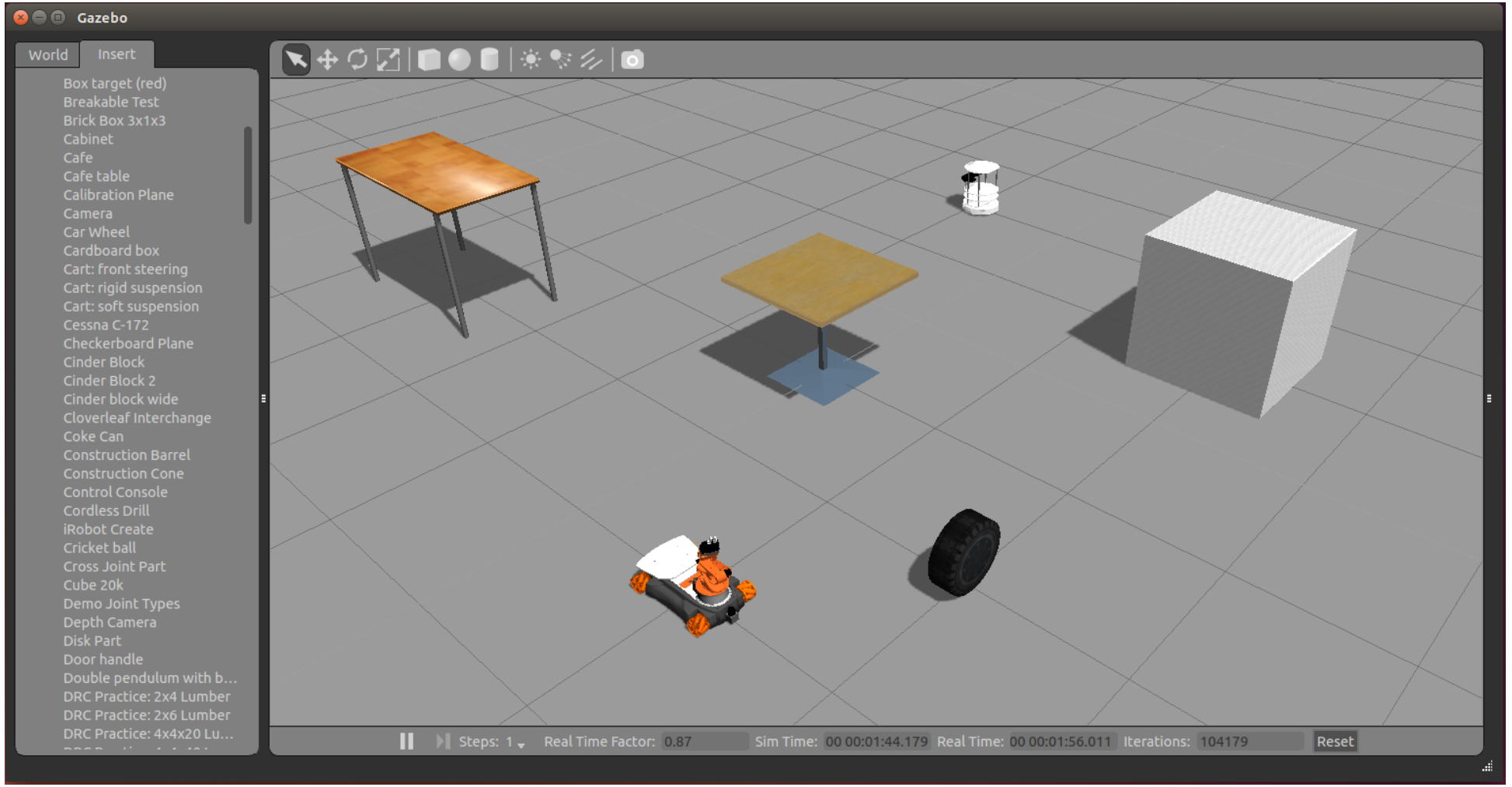
How-world model



- **Gazebo** - 使用SDF（Simulation Description Format）仿真描述格式定义仿真环境和模型。

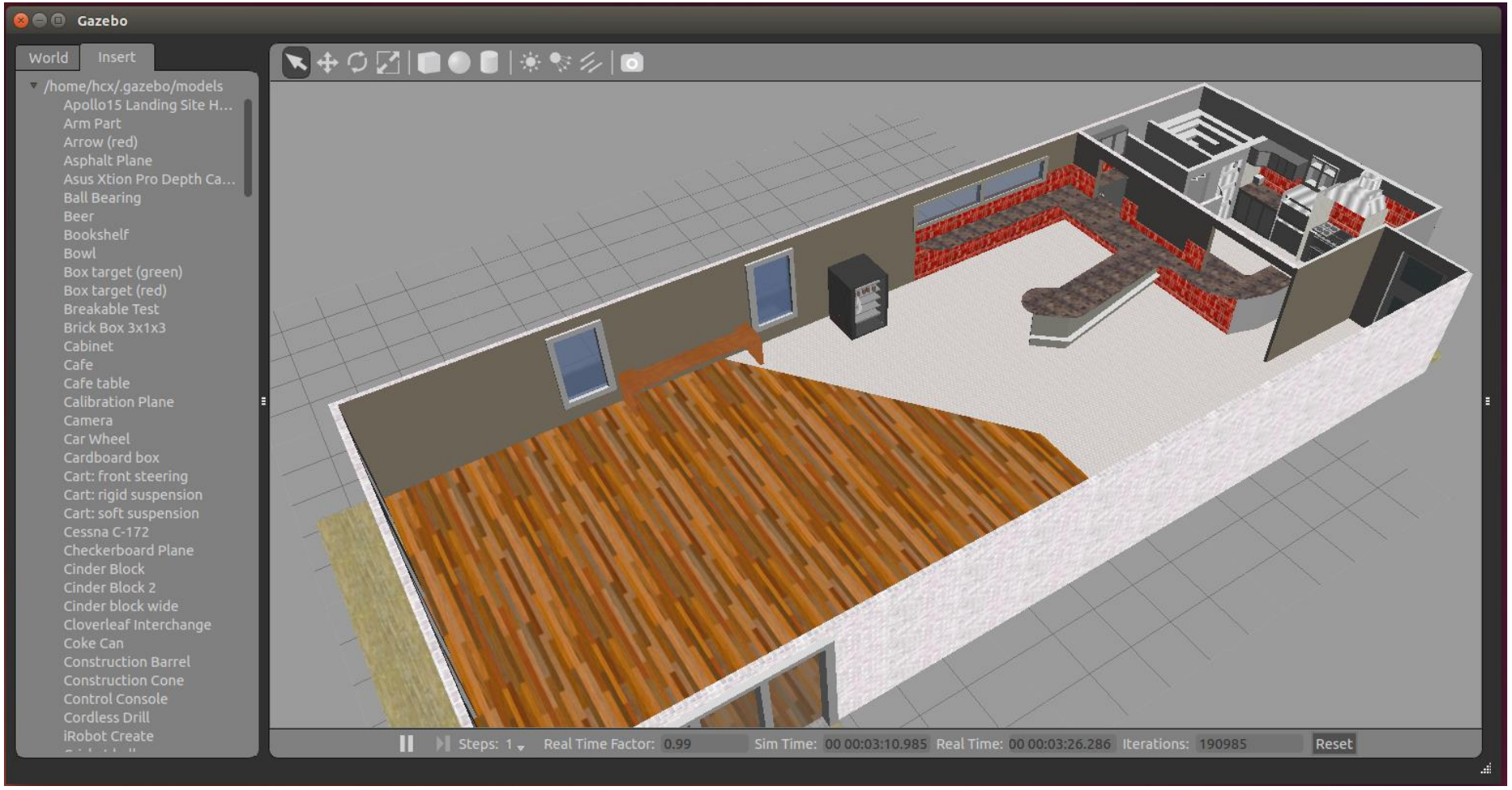
- **ROS** - 使用URDF（Universal Robotic Description Format）通用机器人描述格式来定义机器人模型，且此格式定义的文件不能直接用于Gazebo，使用中需加以转换，添加用于描述仿真定义的标签。

How-创建仿真环境



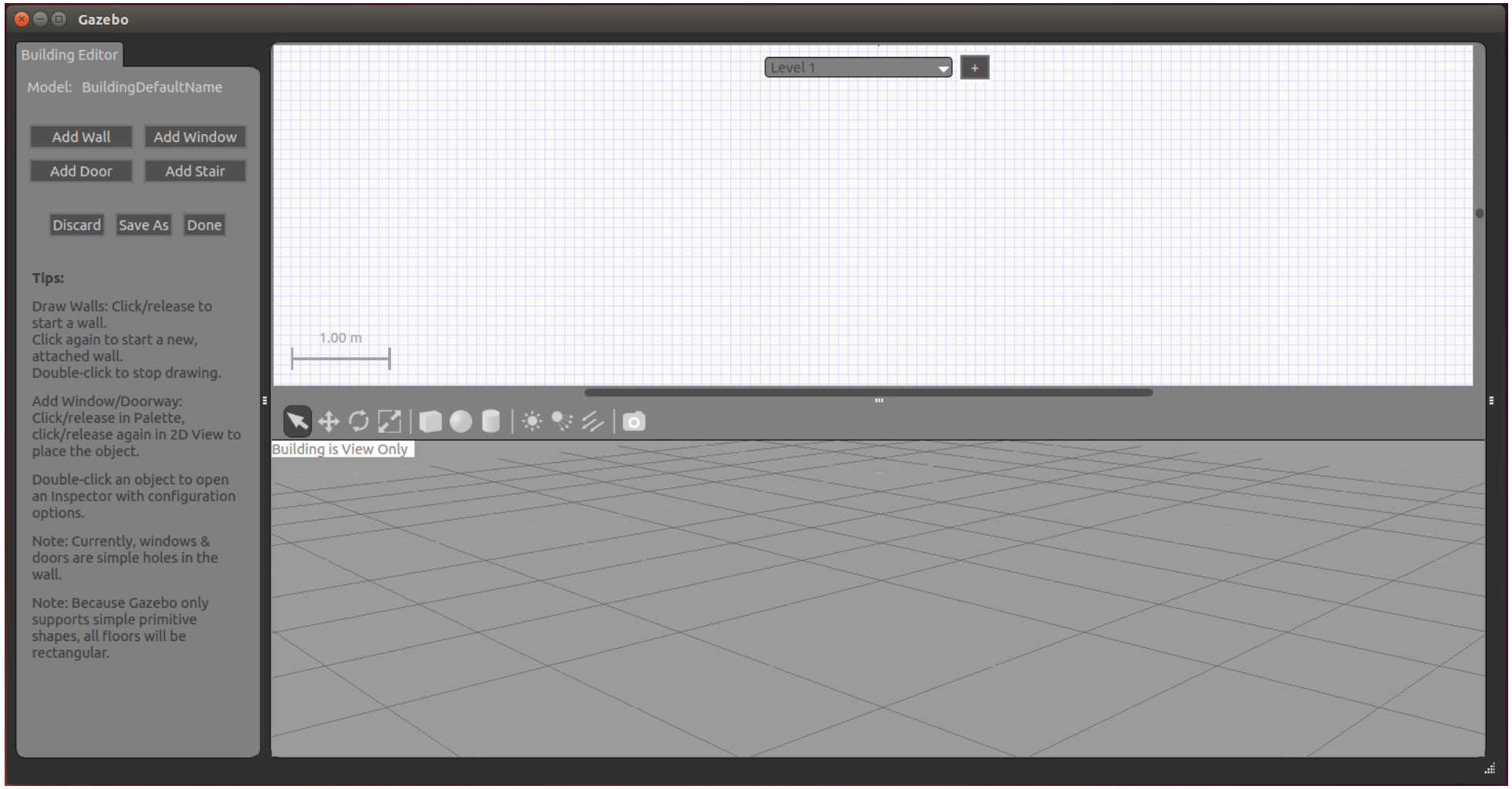
直接添加环境模型

How-创建仿真环境



直接添加环境模型

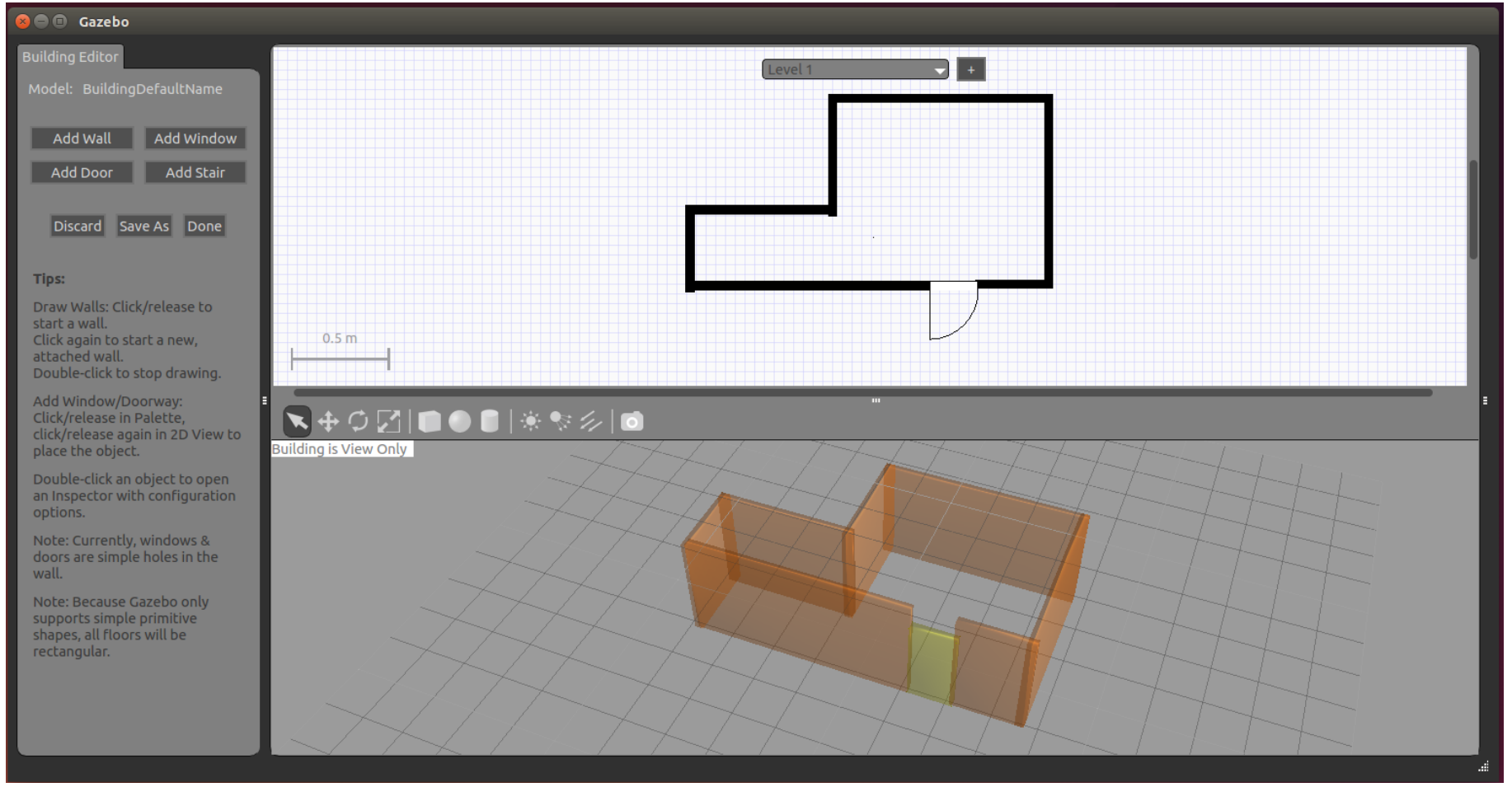
How-创建仿真环境



使用Building Editor

星河智能 Algalaxy

How-创建仿真环境



使用Building Editor

星河智能 Algalaxy

How-模型修改

➢ 必须步骤

- ✓ 在每一个<link>内添加<inertia>标签。
- ✓ 向URDF文件中的joint添加<transmission>标签
- ✓ 向URDF文件中添加ros_control插件，并添加相应插件的controller配置文件

➢ 可选步骤

- ✓ 在每一个<link>内添加<gazebo>标签。将外观颜色转换成Gazebo格式；将stl文件转换成dae文件，获得更好的渲染效果；添加传感器插件。
- ✓ 在每一个<joint>内添加<gazebo>标签。设置适当的阻尼动力系数；添加执行器控制插件
- ✓ 在<robot>标签内添加<gazebo>

How-模型修改

```
<!-- Macro for inertia matrix -->
<xacro:macro name="sphere_inertial_matrix" params="m r">
    <inertial>
        <mass value="${m}" />
        <inertia ixx="${2*m*r*r/5}" ixy="0" ixz="0"
            iyy="${2*m*r*r/5}" iyz="0"
            izz="${2*m*r*r/5}" />
    </inertial>
</xacro:macro>

<xacro:macro name="cylinder_inertial_matrix" params="m r h">
    <inertial>
        <mass value="${m}" />
        <inertia ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
            iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
            izz="${m*r*r/2}" />
    </inertial>
</xacro:macro>

<xacro:macro name="box_inertial_matrix" params="m w h d">
    <inertial>
        <mass value="${m}" />
        <inertia ixx="${m*(h*h+d*d)/12}" ixy = "0" ixz = "0"
            iyy="${m*(w*w+d*d)/12}" iyz = "0"
            izz="${m*(w*w+h*h)/12}" />
    </inertial>
</xacro:macro>
```

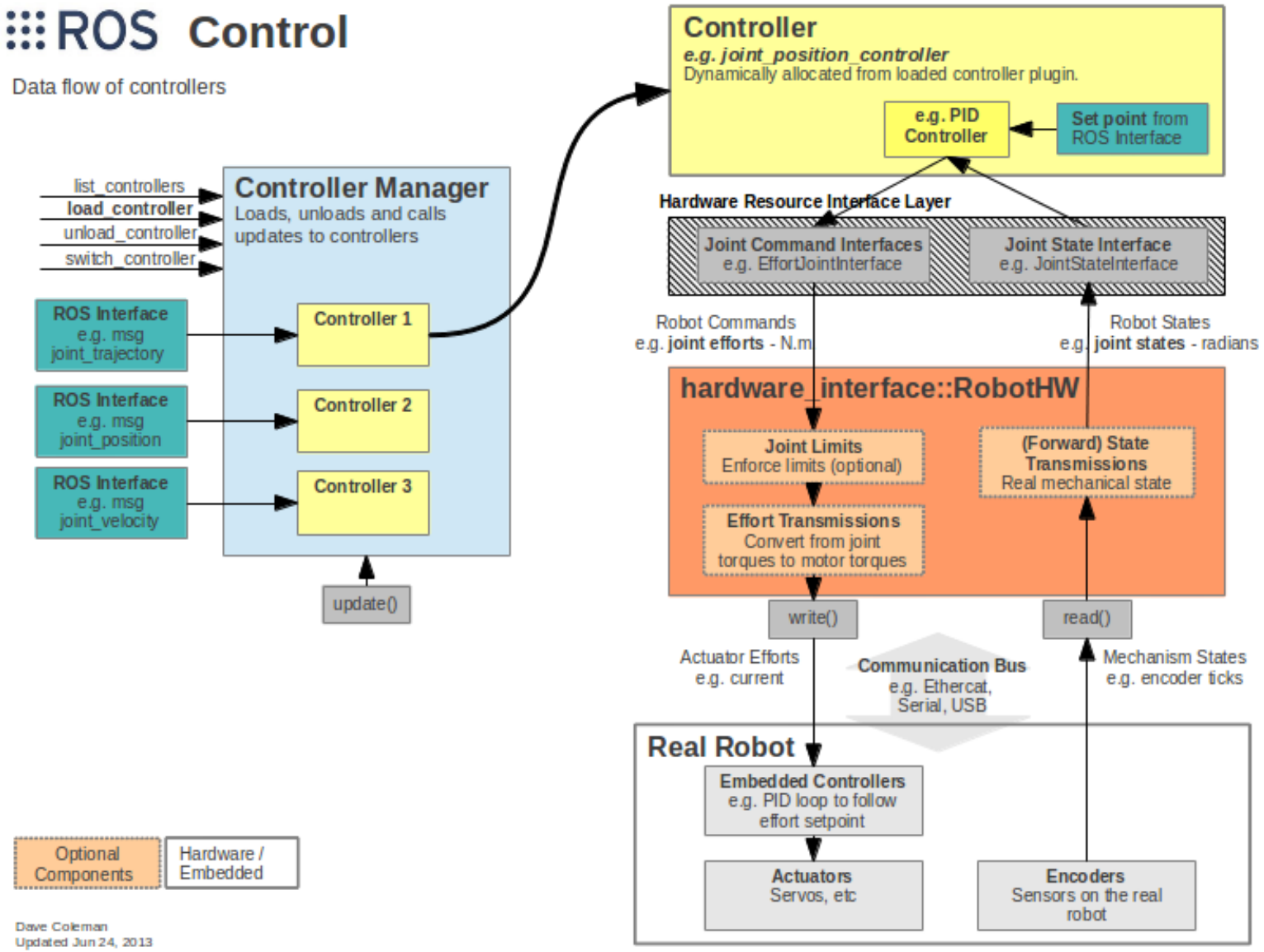
添加<inertia>标签

How-模型修改

```
<!-- Transmission is important to link the joints and the controller -->
<transmission name="wheel_${lr}_joint_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="base_to_wheel_${lr}_joint">
    <hardwareInterface>VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="wheel_${lr}_joint_motor">
    <hardwareInterface>VelocityJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

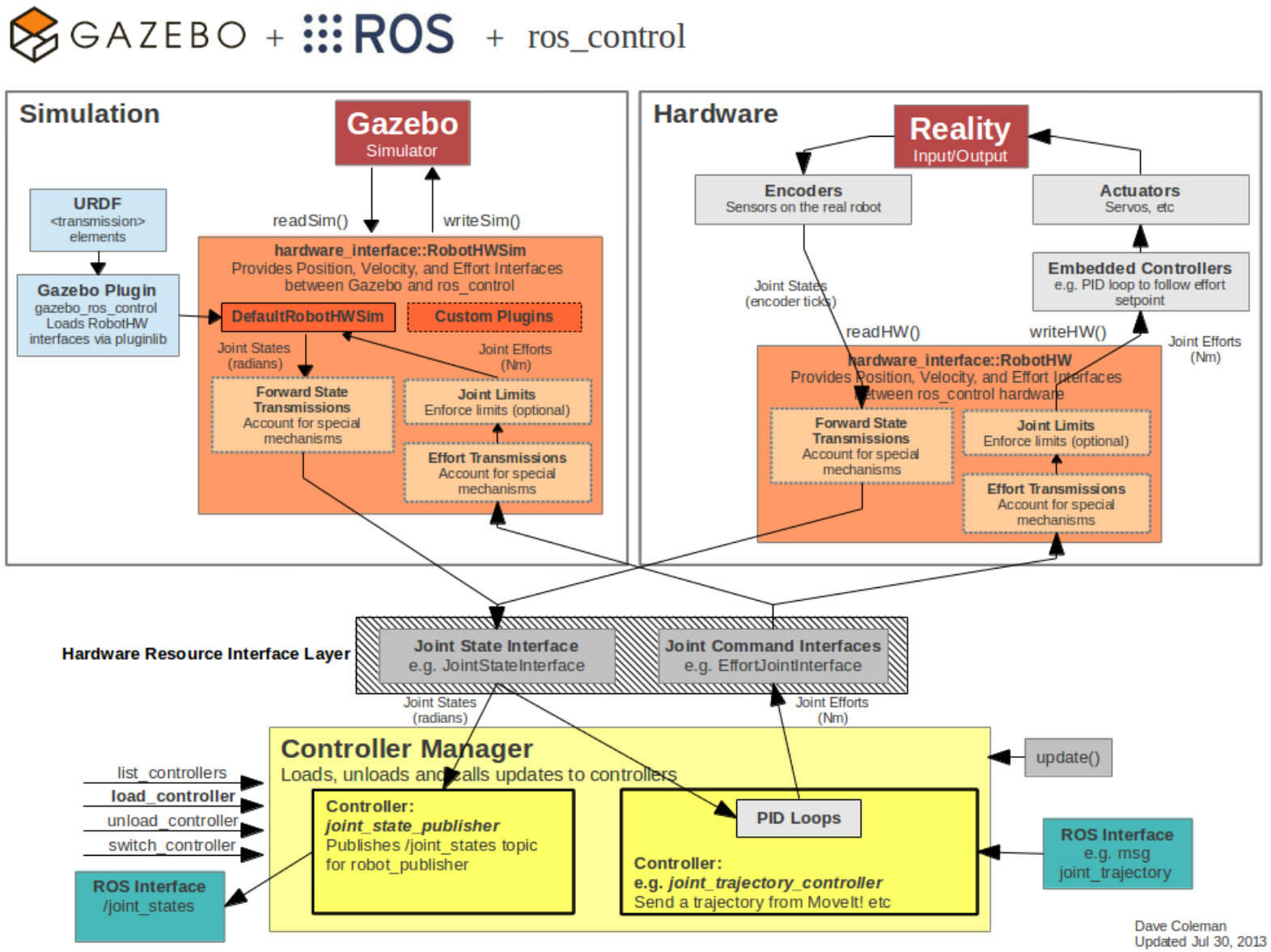
添加<transmission>标签

How-ros_control



- **ros_control**
  - ✓ 硬件接口
  - ✓ 控制器接口
  - ✓ 控制器管理
  - ✓ Transmissions
  - ✓ 控制工具箱

How-ros_control



How-进行仿真

URDF文件

```
<!-- ros control plugin  -->
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>
</gazebo>
```

controller配置文件

```
litebot_controller:
    type: "diff_drive_controller/DiffDriveController"
    left_wheel: "base_to_wheel_left_joint"
    right_wheel: "base_to_wheel_right_joint"
    wheel_separation: 0.2
    wheel_radius: 0.04
    publish_rate: 50.0 # defaults to 50
    pose_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
    twist_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
    cmd_vel_timeout: 20.0 # we test this separately, give plenty for the other tests
```

lanuch文件

```
<!-- Load joint controller configurations from YAML file to parameter server -->
<rosparam file="$(find litebot_description)/config/controller.yaml" command="load"/>

<!-- load the controllers -->
<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
  output="screen" args="litebot_controller"/>
```

添加ros_control插件

How-进行仿真

URDF文件

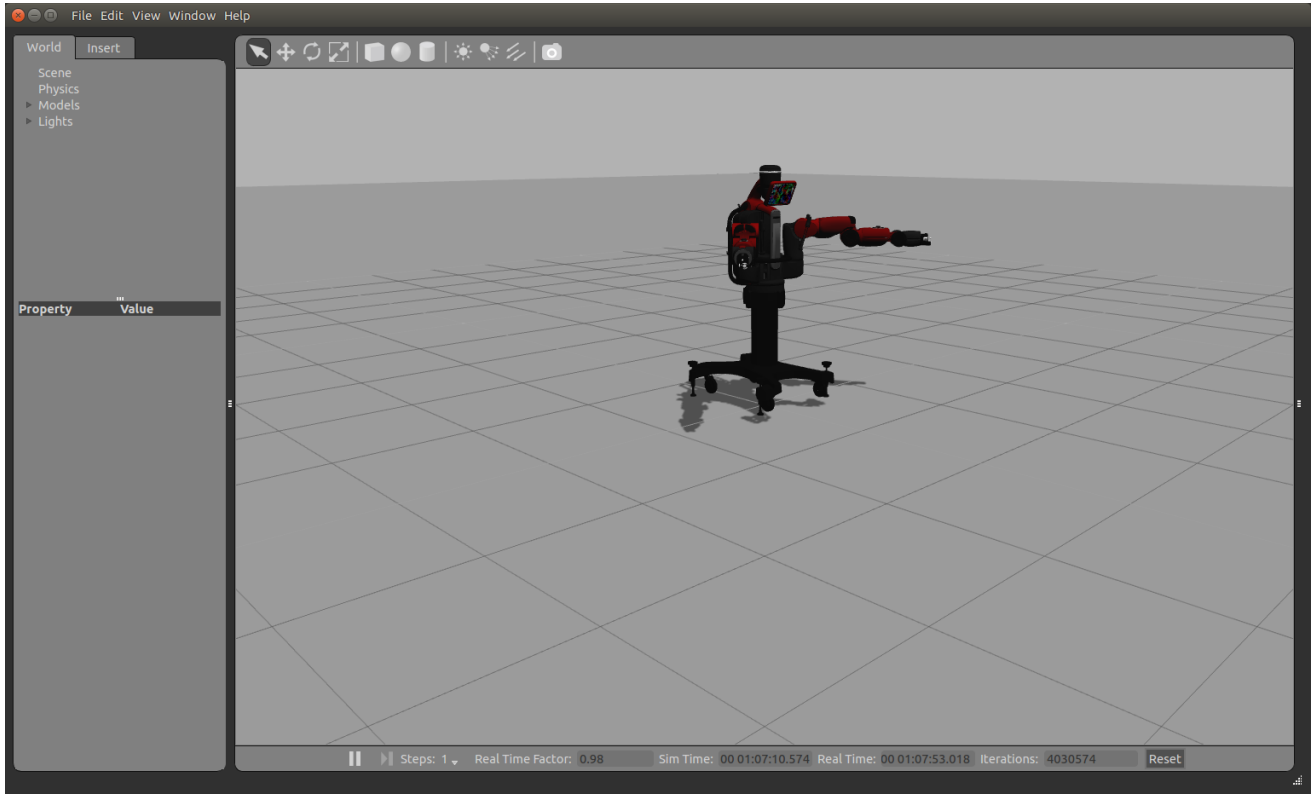
添加传感器插件

```
<gazebo reference="${prefix}_link">
  <sensor type="depth" name="${prefix}">
    <always_on>true</always_on>
    <update_rate>20.0</update_rate>
    <camera>
      <horizontal_fov>${60.0*M_PI/180.0}</horizontal_fov>
      <image>
        <format>R8G8B8</format>
        <width>640</width>
        <height>480</height>
      </image>
      <clip>
        <near>0.05</near>
        <far>8.0</far>
      </clip>
    </camera>
    <plugin name="kinect_${prefix}_controller" filename="libgazebo_ros_openni_kinect.so">
      <cameraName>${prefix}</cameraName>
      <alwaysOn>true</alwaysOn>
      <updateRate>10</updateRate>
      <imageTopicName>rgb/image_raw</imageTopicName>
      <depthImageTopicName>depth/image_raw</depthImageTopicName>
      <pointCloudTopicName>depth/points</pointCloudTopicName>
      <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
      <depthImageCameraInfoTopicName>depth/camera_info</depthImageCameraInfoTopicName>
      <frameName>${prefix}_frame_optical</frameName>
      <baseline>0.1</baseline>
      <distortion_k1>0.0</distortion_k1>
      <distortion_k2>0.0</distortion_k2>
      <distortion_k3>0.0</distortion_k3>
      <distortion_t1>0.0</distortion_t1>
      <distortion_t2>0.0</distortion_t2>
      <pointCloudCutoff>0.4</pointCloudCutoff>
    </plugin>
  </sensor>
</gazebo>
```

How-模型加载

加载 ⇨ 用gazebo_ros功能包下名为*spawn_model*的python脚本，通过封装接口向Gazebo发起服务调用，请求加载指定的URDF文件。

```
rosrun gazebo_ros spawn_model -file `rospack find baxter_description`/urdf/baxter.urdf -urdf -z 1 -model baxter
```



How-模型加载

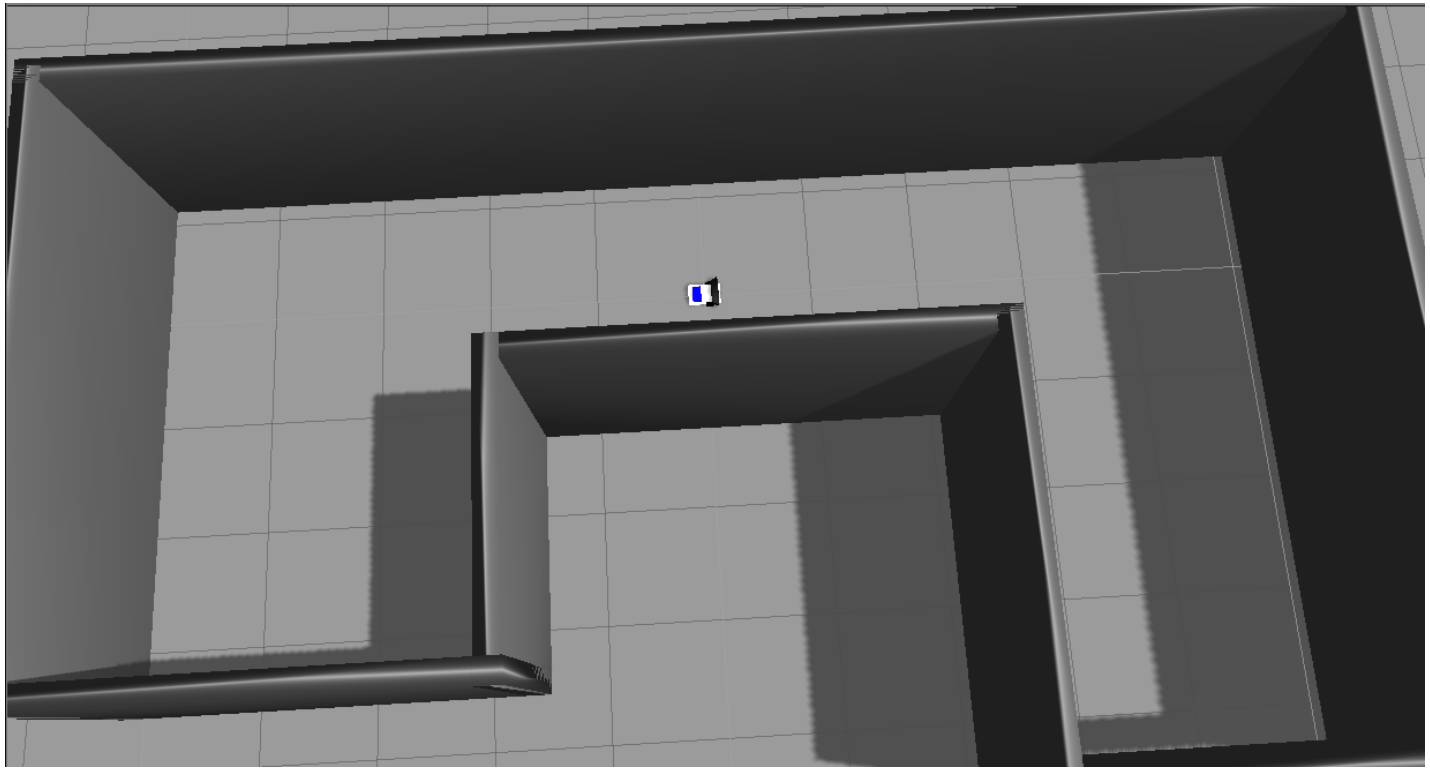
对于xacro格式的机器人模型，需要先进行转换成URDF再加载

```
<launch>

  <!-- urdf xml robot description loaded on the Parameter Server-->
  <param name="robot_description" command="$(find xacro)/xacro.py
                '$(find litebot_description)/urdf/litebot_with_kinect_gazebo.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 litebot -param robot_description"/>

</launch>
```

How-模型加载

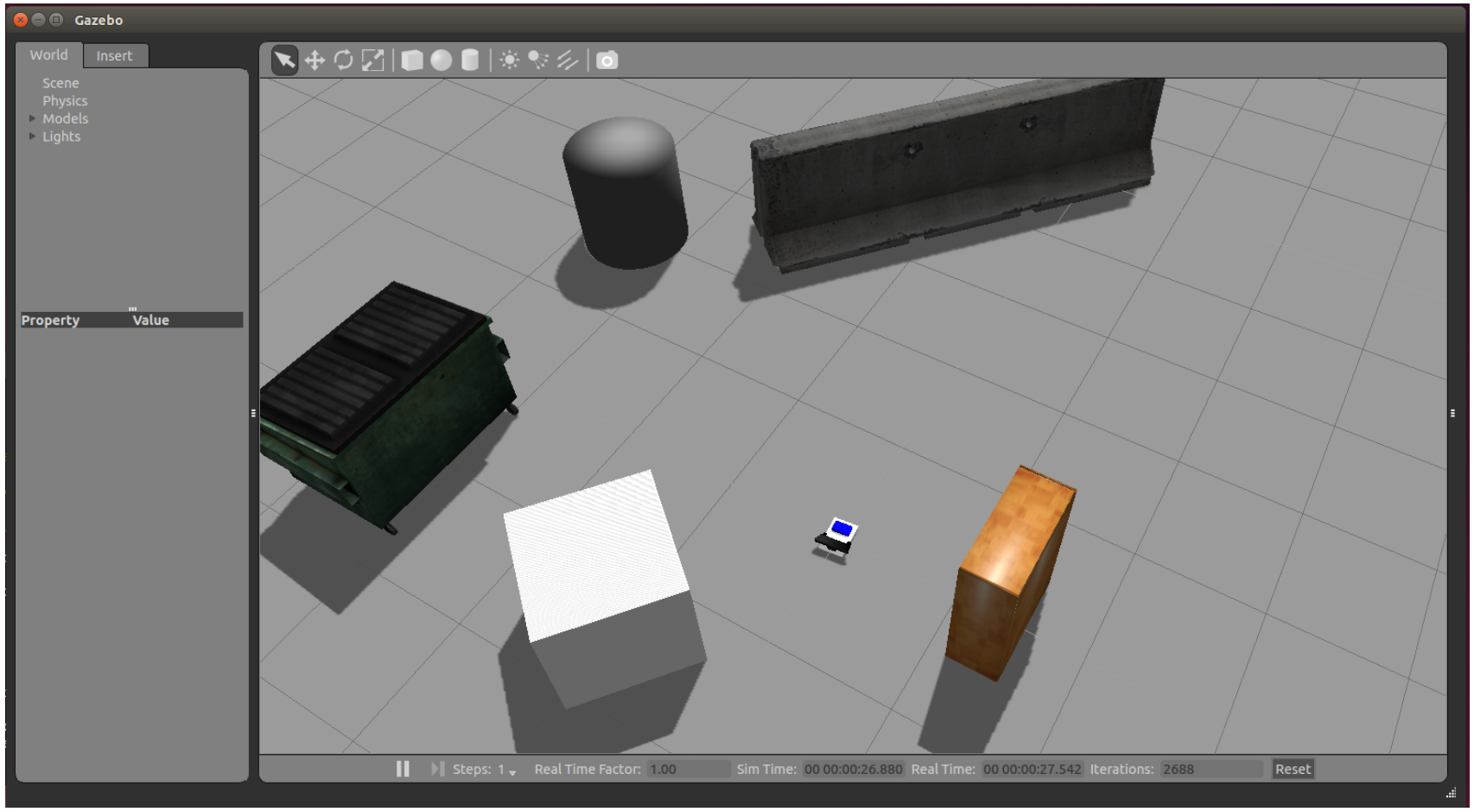
roslaunch litebot_description gazebo_room_world.launch
roslaunch litebot_description add_litebot_to_room.launch

```
spawn_model script started
[INFO] [WallTime: 1500134881.816380] [0.000000] Loading model xml from ros parameter
[INFO] [WallTime: 1500134881.820322] [0.000000] Waiting for service /gazebo/spawn_urdf_model
[INFO] [WallTime: 1500134881.822726] [0.000000] Calling service /gazebo/spawn_urdf_model
[INFO] [WallTime: 1500134882.348089] [63.220000] Spawn status: SpawnModel: Successfully spawned model
```



How-进行仿真

```
roslaunch litebot_description view_litebot_with_kinect_gazebo.launch
rosrun litebot_description litebot_teleop.py
```

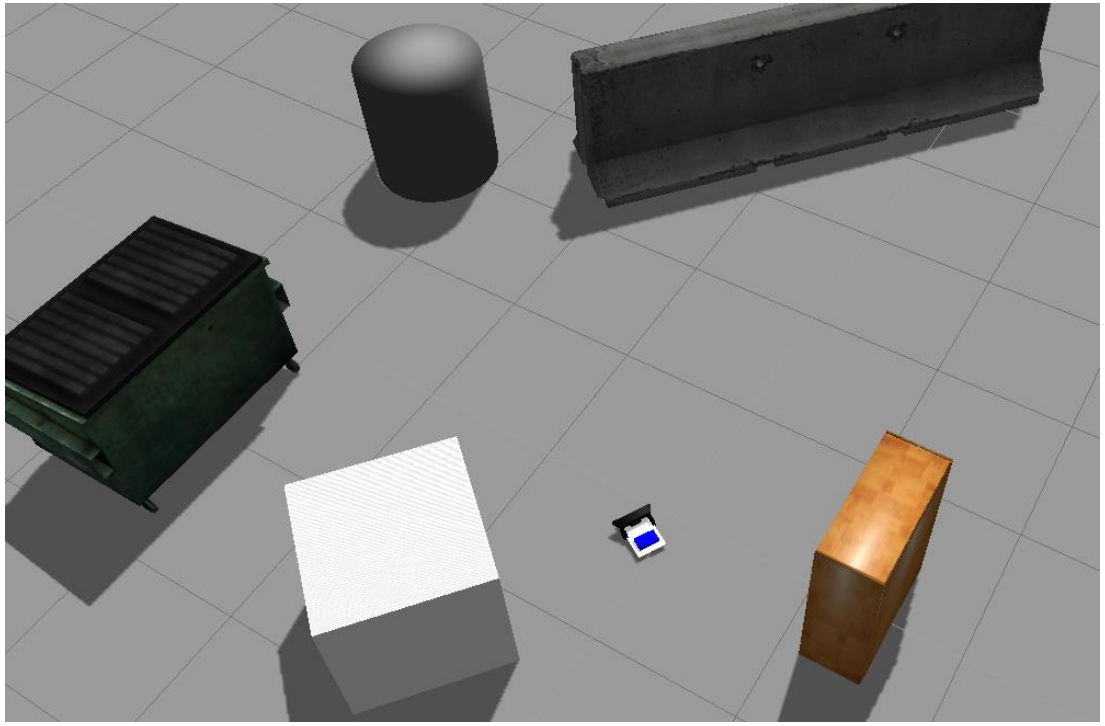


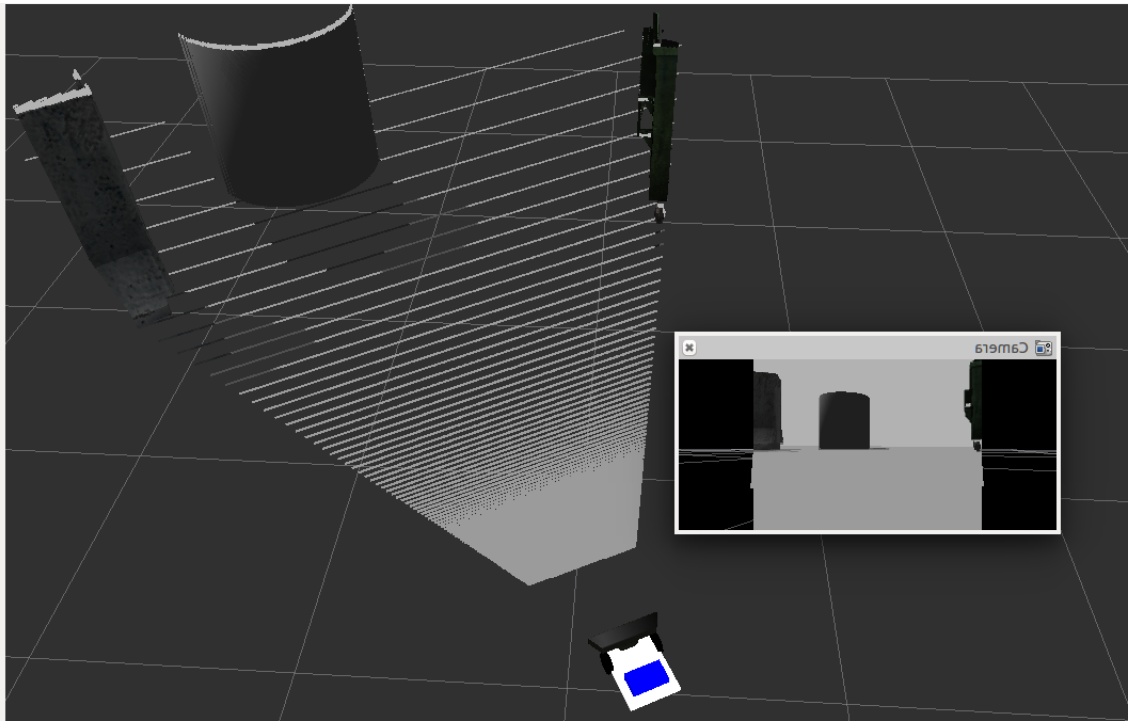
How-进行仿真

```
➜  ~ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/odom
/rosout
/rosout_agg
/tf
/tf_static
```

查看系统中的消息，可以看到传感器的话题和机器人速度控制的话题

How-进行仿真





配合rviz进行仿真

How-机器人仿真的步骤

创建仿真环境

- 在gazebo中直接添加环境模型
- 使用gazebo building editor绘制环境模型

修改机器人模型

- 在机器人模型中加载ros control和传感器插件
- 将机器人模型加载到gazebo中

开始仿真

- 配合rviz等工具进行仿真

总结 Summarize

- gazebo是一款开源的机器人物理仿真平台。
- 与ROS集成度好，可以利用ROS社区中丰富的资源。
- 仿真步骤：创建环境、修改模型、进行仿真

延展阅读

- ✓ http://www.gazebosim.org
- ✓ http://www.gazebosim.org/tutorials
- ✓ http://answers.gazebosim.org
- ✓ http://www.guyuehome.com

Thank you

星河智能
AIgalaxy