Back (/tutorials?cat=install)

URDF in Gazebo

Tutorial: Using a URDF in Gazebo

The Universal Robotic Description Format (http://www.ros.org/wiki/urdf) (URDF) is an XML file format used in ROS to describe all elements of a robot. To use a URDF file in Gazebo, some additional simulation-specific tags must be added to work properly with Gazebo. This tutorial explains the necessary steps to successfully use your URDF-based robot in Gazebo, saving you from having to create a separate SDF file from scratch and duplicating description formats. Under the hood, Gazebo will then convert the URDF to SDF automatically.

Background

While URDFs are a useful and standardized format in ROS, they are lacking many features and have not been updated to deal with the evolving needs of robotics. URDF can only specify the kinematic and dynamic properties of a single robot in isolation. URDF can not specify the pose of the robot itself within a world. It is also not a universal description format since it cannot specify joint loops (parallel linkages), and it lacks friction and other properties. Additionally, it cannot specify things that are not robots, such as lights, heightmaps, etc.

On the implementation side, the URDF syntax breaks proper formatting with heavy use of XML attributes, which in turn makes URDF more inflexible. There is also no mechanism for backward compatibility.

To deal with this issue, a new format called the Simulation Description Format (http://sdformat.org) (SDF) was created for use in Gazebo to solve the shortcomings of URDF. SDF is a complete description for everything from the world level down to the robot level. It is scalable, and makes it easy to add and modify elements. The SDF format is itself described using XML, which facilitates a simple upgrade tool to migrate old versions to new versions. It is also self-descriptive.

It is the intention of this author to make URDFs as fully documented and supported in Gazebo as possible, but it is relevant to the reader to understand why the two formats exist and the shortcomings of both. It would be nice if more work was put into URDFs to update them to the current needs of robotics.

Overview of Converting to Gazebo

There are several steps to get a URDF robot properly working in Gazebo. The following is an overview of steps, which are then elaborated on in the rest of this tutorial:

Required

- An `<inertia>` element within each `<link>` element must be properly specified and configured.

Optional

- Add a `<gazebo>` element for every `<link>`
  - Convert visual colors to Gazebo format
  - Convert stl files to dae files for better textures
  - Add sensor plugins
- Add a `<gazebo>` element for every `<joint>`
  - Set proper damping dynamics
  - Add actuator control plugins
- Add a `<gazebo>` element for the `<robot>` element
- Add a `<link name="world"/>` link if the robot should be rigidly attached to the world/base_link

The `<gazebo>` Element

The `<gazebo>` element is an extension to the URDF used for specifying additional properties needed for simulation purposes in Gazebo. It allows you to specify the properties found in the SDF format that are not included in the URDF format. None of the elements within a `<gazebo>` element are required because default values will be automatically included. There are three different types of `<gazebo>` elements - one for the `<robot>` tag, one for `<link>` tags, and one for `<joint>` tags. We will discuss the attributes and elements within each type of `<gazebo>` element throughout this tutorial.

Prerequisites

The first step to getting your robot working in Gazebo is to have a working URDF file from the corresponding ROS URDF Tutorials (http://www.ros.org/wiki/urdf/Tutorials). Test your URDF by viewing it in Rviz (http://www.ros.org/wiki/rviz) before proceeding to configure your robot with Gazebo. In this tutorial, we'll use a simple demo robot named RRBot. Feel free to follow along with this robot or your own bot.

Getting RRBot

RRBot, or "Revolute-Revolute Manipulator Robot", is a simple 3-linkage, 2-joint arm that we will use to demonstrate various features of Gazebo and URDFs. It essentially a double inverted pendulum (http://en.wikipedia.org/wiki/Double_inverted_pendulum) and demonstrates some fun control concepts within a simulator.

To get RRBot, clone the gazebo_ros_demos Github repo (https://github.com/ros-simulation/gazebo_ros_demos.git) into the `/src` folder of your catkin workspace and rebuild your workspace:

```
cd ~/catkin_ws/src/
git clone https://github.com/ros-simulation/gazebo_ros_demos.git
cd ..
catkin_make
```

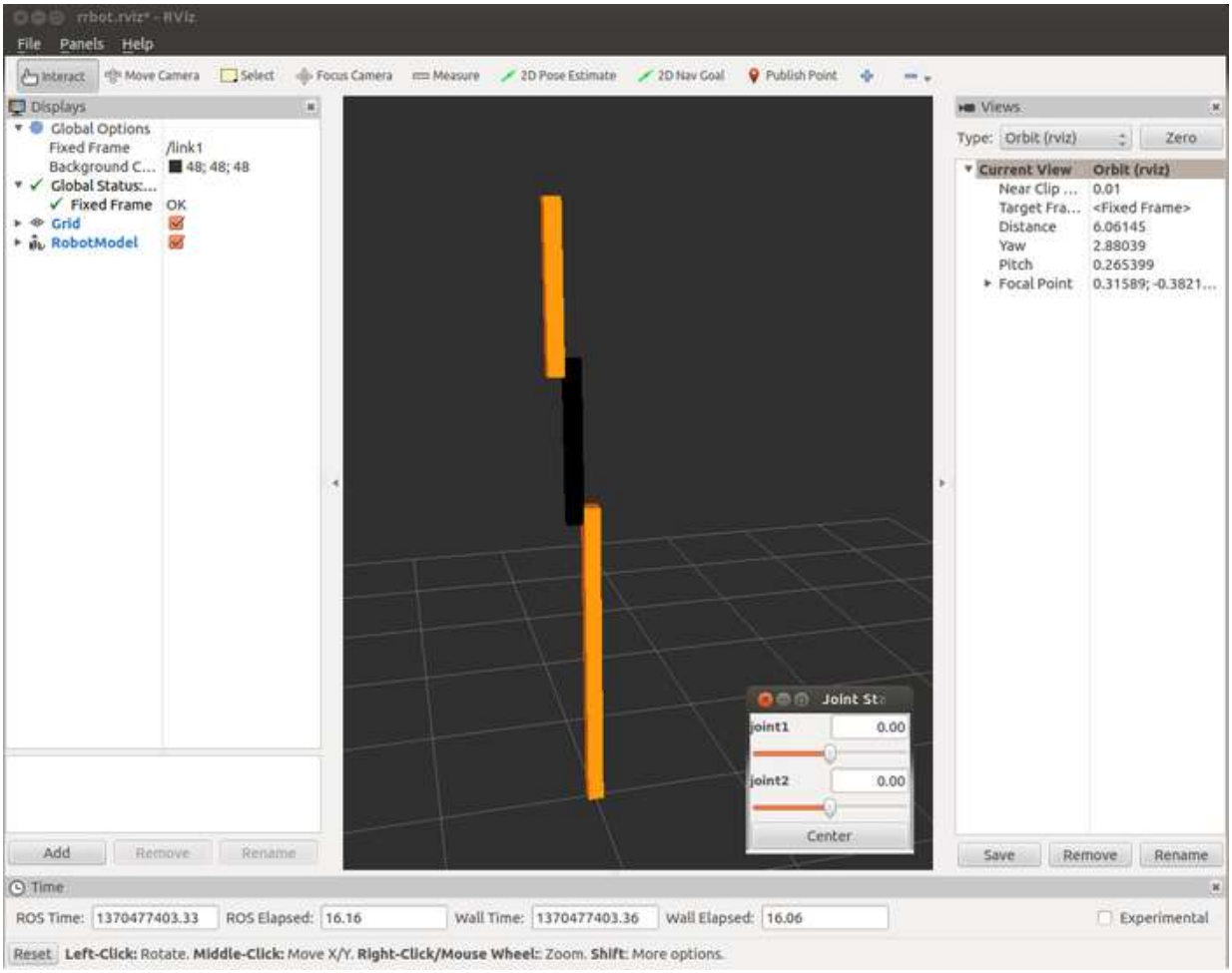
If any of this is unfamiliar, be sure you have read the previous ROS Overview Tutorials (http://gazebosim.org/tutorials?tut=ros_overview).

View in Rviz

To check if everything is working, launch RRBot in Rviz:

```
roslaunch rrbot_description rrbot_rviz.launch
```

And you should see our little bot like so:



If you do not get this, try killing all old roscore processes with `killall roscore` and relaunching RViz.

You should also be able to play with the slider bars in the Joint State Publisher window to move the two joints.

It is important that while converting your robot to work in Gazebo, you don't break Rviz or other ROS-application functionality, so its nice to occasionally test your robot in Rviz to make sure everything still works.

The gazebo_ros_control (http://gazebosim.org/tutorials?tut=ros_control) tutorial will explain how to use Rviz to monitor the state of your simulated robot by publishing `/joint_states` directly from Gazebo. In the previous example, the RRBot in Rviz is getting its `/joint_states` from a fake `joint_states_publisher` node (the window with the slider bars).

Examine the RRBot URDF

The rest of this tutorial will refer to various aspects of the RRBot URDF. Go ahead and view the rrbot.xacro file (https://github.com/ros-simulation/gazebo_ros_demos/blob/master/rrbot_description/urdf/rrbot.xacro) now:

```
rosed rrbot_description rrbot.xacro
```

Note that we are using Xacro (http://ros.org/wiki/xacro) to make some of the link and joint calculations easier. We are also including two additional files:

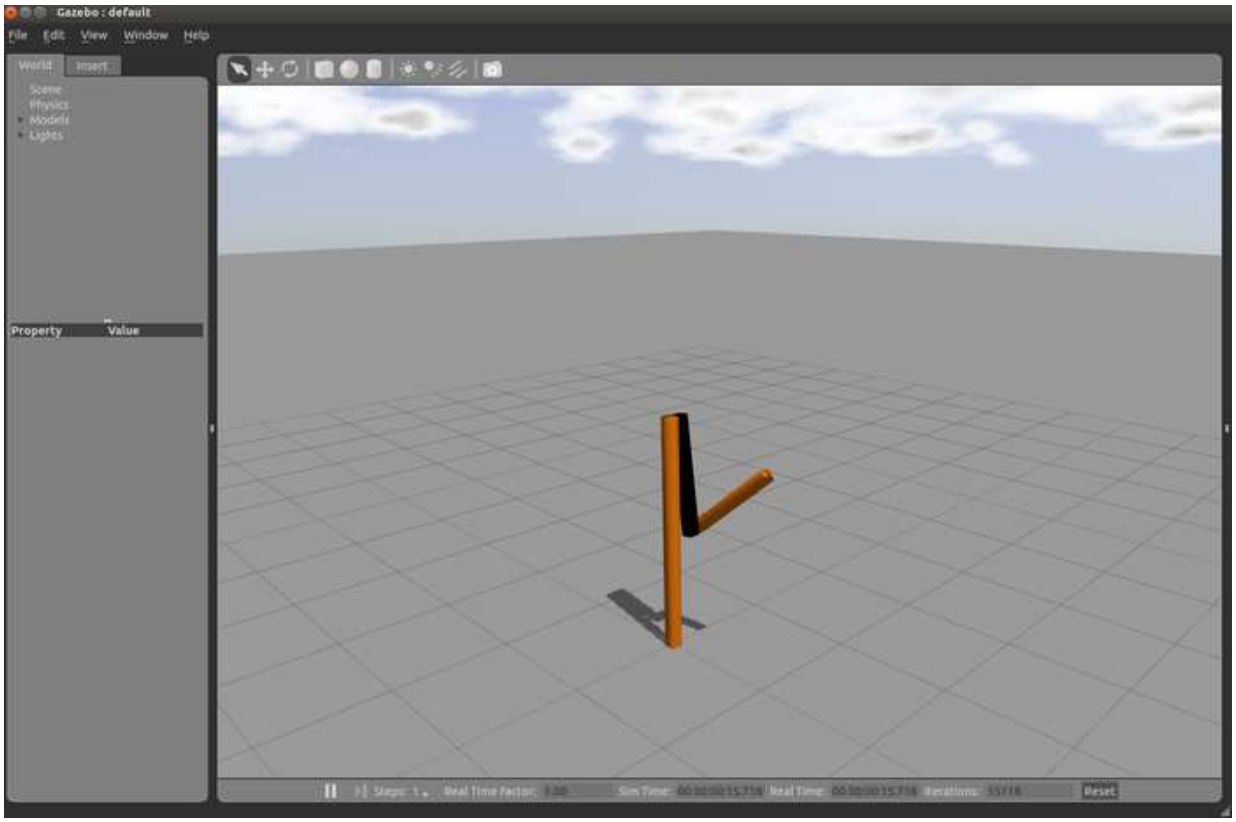
- rrbot.gazebo (https://github.com/ros-simulation/gazebo_ros_demos/blob/master/rrbot_description/urdf/rrbot.gazebo) a Gazebo specific file that includes most of our Gazebo-specific XML elements including the tags
- materials.xacro (https://github.com/ros-simulation/gazebo_ros_demos/blob/master/rrbot_description/urdf/materials.xacro) a simple Rviz colors file for storing rgba values, not really necessary but a nice convention

View in Gazebo

You should also be able to launch RRBot into Gazebo:

```
roslaunch rrbot_gazebo rrbot_world.launch
```

In the launched Gazebo window you should see the robot standing straight up. Despite there being no intentional disturbances in the physics simulator by default, numerical errors should start to build up and cause the double inverted pendulum to fall after a few seconds. The following is a mid-swing screenshot of the RRBot:



Eventually the arm should come to a complete stop. We encourage you to tweak and test various aspects of the URDF during the following tutorials to help you learn more about simulating URDF robots.

Header of a URDF File

There have been many API changes in Gazebo and the required URDF format, one of which that Gazebo xml-schema namespaces are no longer needed. If your URDF has something like:

```
<robot xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
       xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
       xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
       xmlns:xacro="http://playerstage.sourceforge.net/gazebo/xmlschema/#xacro"
       name="pr2" >
```

You can remove them. All you need in your root element tag is the name of the robot and optionally the xml namespace for xacro if you are using that:

```
<robot name="rrbot" xmlns:xacro="http://www.ros.org/wiki/xacro">
```

`<gazebo>` element for the tag

If a `<gazebo>` element is used without a `reference=""` property, it is assumed the `<gazebo>` element is for the whole robot model. The elements for a `<robot>` inside the `<gazebo>` tag are listed in the following table:

Name	Type	Description

static	bool	If set to true, the model is immovable. Otherwise the model is simulated in the dynamics engine.

Elements within a `<gazebo>` tag that are not in the above table are directly inserted into the SDF `<model>` tag for the generated SDF. This is particularly useful for plugins, as discussed in the ROS Motor and Sensor Plugins (http://gazebosim.org/tutorials?tut=ros_gzplugins) tutorial.

Rigidly Fixing A Model to the World

If you would like your URDF model to be permanently attached to the world frame (the ground plane), you must create a "world" link and a joint that fixes it to the base of your model. RRBot accomplishes this with the following:

```
<!-- Used for fixing robot to Gazebo 'base_link' -->
<link name="world"/>

<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="link1"/>
</joint>
```

If however you have a mobile base or some other moving robot, you do not need this link or joint.

Links

Be sure you are familiar with the URDF link (http://www.ros.org/wiki/urdf/XML/link) element.

The following is an example link from RRBot:

```
<!-- Base Link -->
<link name="link1">
  <collision>
    <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
    <geometry>
      <box size="${width} ${width} ${height1}"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
    <geometry>
      <box size="${width} ${width} ${height1}"/>
    </geometry>
    <material name="orange"/>
  </visual>

  <inertial>
    <origin xyz="0 0 1" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia
      ixx="1.0" ixy="0.0" ixz="0.0"
      iyy="1.0" iyz="0.0"
      izz="1.0"/>
  </inertial>
</link>
```

Note On Units

As per ROS REP 103: Standard Units of measure and Coordinate Conventions (http://www.ros.org/reps/rep-0103.html), units in Gazebo should be specified in meters and kilograms. Gazebo could possibly be used with imperial units if the constants such as gravity were changed manually, but by default gravity is 9.81 m/s^2. When specifying mass, use units of kilograms.

`<collision>` and `<visual>` elements

These tags work essentially the same in Gazebo as in Rviz. It is important that you specify both though, because unlike some ROS applications, Gazebo will not use your `<visual>` elements as `<collision>` elements if you do not explicitly specify a `<collision>` element. Instead, Gazebo will treat your link as "invisible" to laser scanners and collision checking.

Simplify collision model

You can use the same geometry or meshes for both your collision and visual elements, though for performance improvements we strongly suggest you have a simplified model/meshes for your collision geometry. A good open-source tool for simplifying meshes is Blender. There are many closed-source tools, such as Maya and 3DS Max, which can also simplify meshes.

Materials: Using proper colors and textures

A standard URDF can specify colors using a tag such as in the RRBot:

```
<material name="orange"/>
```

With the color orange defined separately such as in the file materials.xacro (https://github.com/ros-simulation/gazebo_ros_demos/blob/master/rrbot_description/urdf/materials.xacro#L20-L22):

```
<material name="orange">
  <color rgba="${255/255} ${108/255} ${10/255} 1.0"/>
</material>
```

Unfortunately, this method of specifying link colors does not work in Gazebo as it adopts OGRE's material scripts for coloring and texturing links. Instead, a Gazebo material tag must be specified for each link, such as:

```
<gazebo reference="link1">
  <material>Gazebo/Orange</material>
</gazebo>
```

As mentioned earlier, in the RRBot example we have chosen to include all Gazebo-specific tag in a secondary file called rrbot.gazebo (https://github.com/ros-simulation/gazebo_ros_demos/blob/master/rrbot_description/urdf/rrbot.gazebo). You can find the `<link>` and `<material>` elements there.

The default available materials in Gazebo can be found in the Gazebo source code at gazebo/media/materials/scripts/gazebo.material (https://bitbucket.org/osrf/gazebo/src/default/media/materials/scripts/gazebo.material).

For more advanced or custom materials, you can create your own OGRE colors or textures. See:

- The `<material>` SDF documentation (http://sdformat.org/spec?ver=1.5&elem=material)
- OGRE materials documentation (http://www.ogre3d.org/tikiwiki/Materials)

STL and Collada files

Like in Rviz, Gazebo can use both STL (http://en.wikipedia.org/wiki/STL_(file_format)) and Collada (http://en.wikipedia.org/wiki/Collada) files. It is generally recommended you use Collada (.dae) files because they support colors and textures, whereas with STL files you can only have a solidly colored link.

`<inertial>` Element

For the Gazebo physics engine to work properly, the `<inertial>` element must be provided as documented on the URDF link element (http://www.ros.org/wiki/urdf/XML/link) page. For links to not be ignored in Gazebo, their mass must be greater than zero. Additionally, links with zero principal moment of inertia (ixx, iyy, izz) could lead to infinite acceleration under any finite torque application.

Determining the correct values for each link is required to get accurate physics approximations in Gazebo. This can be performed by conducting various measurements of the robots parts, or by using CAD software like Solidworks that includes features for approximating these values. For beginners, you can also just make the values up.

An example inertia element from the RRBot first link:

```
<inertial>
  <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
  <mass value="1"/>
  <inertia
    ixx="1.0" ixy="0.0" ixz="0.0"
    iyy="1.0" iyz="0.0"
    izz="1.0"/>
</inertial>
```

The origin tag represents the center of mass of this link. By setting the center of mass to half the height of the RRBot's rectangular link, we center the mass in the middle. You can visually check if your center of mass is correct in your URDF within Gazebo by clicking on the "View" menu of Gazebo and selecting both "Wireframe" and "Center of Mass".

In this example robot, both the mass and inertia matrix are made up values since this robot has no real-world counterpart.

`<gazebo>` Elements For Links

List of elements that are individually parsed:

Name	Type	Description
material	value	Material of visual element
gravity	bool	Use gravity
dampingFactor	double	Exponential velocity decay of the link velocity - takes the value and multiplies the previous link velocity by (1-dampingFactor).
maxVel	double	maximum contact correction velocity truncation term.
minDepth	double	minimum allowable depth before contact correction impulse is applied
mu1	double	Friction coefficients μ for the principal contact directions along the contact surface as defined by the Open Dynamics Engine (ODE) (http://www.ode.org) (see parameter descriptions in ODE's user guide (http://www.ode.org/ode-latest-userguide.html#sec_7_3_7))
mu2		
fdir1	string	3-tuple specifying direction of mu1 in the collision local reference frame.
kp	double	Contact stiffness k_p and damping k_d for rigid body contacts as defined by ODE (ODE uses erp and cfm (http://www.ode.org/ode-latest-userguide.html#sec_7_3_7) but there is a mapping between erp/cfm and stiffness/damping (https://bitbucket.org/osrf/gazebo/src/b4d836c3ab3b0a/gazebo/physics/ode/ODEJoint.cc#ODEJoint.cc-982))
kd		
selfCollide	bool	If true, the link can collide with other links in the model.
maxContacts	int	Maximum number of contacts allowed between two entities. This value overrides the max_contacts element defined in physics.
laserRetro	double	intensity value returned by laser sensor.

Similar to `<gazebo>` elements for `<robot>`, any arbitrary blobs that are not parsed according to the table above are inserted into the the corresponding `<link>` element in the SDF. This is particularly useful for plugins, as discussed in the ROS Motor and Sensor Plugins (http://gazebosim.org/tutorials?tut=ros_gzplugins) tutorial.

RRBot Example of element

In the RRBot, the friction coefficients of the two non-fixed linked were specified so that if a collision occurred more accurate contact interactions were simulated. The following is an example link's `<gazebo>` tag:

```
<gazebo reference="link2">
  <mu1>0.2</mu1>
  <mu2>0.2</mu2>
  <material>Gazebo/Black</material>
</gazebo>
```

Joints

Make sure you are familiar with the URDF joint documentation (http://www.ros.org/wiki/urdf/XML/joint). However, not all of the elements documented for URDF joints are applicable to Gazebo:

- The `<origin>`, `<parent>` and `<child>` are required
- `<calibration>` and `<safety_controller>` are ignored
- In the `<dynamics>` tag, only the `damping` property is used for gazebo4 and earlier. Gazebo5 and up also uses the `friction` property.
- All of properties in the `<limit>` tag are optional

RRBot Example

The following is a joint used in the RRBot:

```
<joint name="joint2" type="continuous">
  <parent link="link2"/>
  <child link="link3"/>
  <origin xyz="0 ${width} ${height2 - axel_offset*2}" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <dynamics damping="0.7"/>
</joint>
```

Notice the dynamics element with a viscous damping coefficient of 0.7 N*m*s/rad, damping is simply the amount of opposing force to any joint velocity (in this case torque per angular velocity) that is used to "slow" a moving joint towards rest.

The value of 0.7 N*m*s/rad was decided on by testing different amounts of damping and watching how "realistic" the swinging pendulum appeared. We encourage you to play with this value now (increase/decrease it) to get a feel for how it affects the physics engine.

`<gazebo>` Elements For Joints

Name	Type	Description
stopCfm	double	Joint stop constraint force mixing (cfm) and error reduction parameter (erp) used by ODE
stopErp		
provideFeedback	bool	Allows joints to publish their wrench data (force-torque) via a Gazebo plugin
implicitSpringDamper	bool	If this flag is set to true, ODE will use ERP and CFM to simulate damping. This is a more stable numerical method for damping than the default damping tag. The cfmDamping element is deprecated and should be changed to implicitSpringDamper.
cfmDamping		
fudgeFactor	double	Scale the excess for in a joint motor at joint limits. Should be between zero and one.

Again, similar to `<gazebo>` elements for `<robot>` and `<link>`, any arbitrary blobs that are not parsed according to the table above are inserted into the the corresponding `<joint>` element in the SDF. This is particularly useful for plugins, as discussed in the ROS Motor and Sensor Plugins (http://gazebosim.org/tutorials?tut=ros_gzplugins) tutorial.

Verifying the Gazebo Model Works

With Gazebo installed, an easy tool exists to check if your URDF can be properly converted into a SDF. Simply run the following command:

```
# gazebo2 and below
gzsdf print MODEL.urdf
# gazebo3 and above
gz sdf -p MODEL.urdf
```

This will show you the SDF that has been generated from your input URDF as well as any warnings about missing information required to generate the SDF.

Note: in Gazebo version 1.9 and greater, some of the debug info has been moved to a log file you can view with:

```
cat ~/.gazebo/gzsdf.log
```

Viewing the URDF In Gazebo

Viewing the RRBot in Gazebo was already covered at the beginning of this tutorial. For your own custom robot, we assume its URDF lives in a ROS package named `MYROBOT_description` in the subfolder `/urdf`. The method to open a URDF from that location into Gazebo using ROS was covered in the previous tutorial, Using roslaunch Files to Spawn Models (http://gazebosim.org/tutorials?tut=ros_roslaunch). If you have not completed that tutorial, do so now.

From that tutorial you should have two ROS packages for your custom robot: `MYROBOT_description` and `MYROBOT_gazebo`. To view your robot and test it in Gazebo, you should be able to now run something like:

```
roslaunch MYROBOT_gazebo MYROBOT.launch
```

This should launch both the Gazebo server and GUI client with your robot automatically launched spawned inside.

Tweaking your model

If your robot model behaves unexpectedly within Gazebo, it is likely because your URDF needs further tuning to accurately represent its physics in Gazebo. See the SDF user guide (http://gazebosim.org/sdf.html) for more info on various properties available in Gazebo, which are also available in the URDF via the `<gazebo>` tag.

Sharing your robot with the world

If you have a common robot that other's might want to use in Gazebo, you are encouraged to add your URDF to the Gazebo Model Database (http://gazebosim.org/tutorials?tut=model_structure&cat=build_robot). It is an online server that Gazebo connects to to pull down models from the internet. Its Mercurial repository is located on Bitbucket (https://bitbucket.org/osrf/gazebo_models). See Gazebo Model Database (http://gazebosim.org/tutorials?tut=model_contrib&cat=build_robot) documentation for how to submit a pull request to have your robot added to the database.

Next steps

You have now learned how to use ROS packages containing URDFs with Gazebo, and how to convert your custom URDF to work in Gazebo. You are now ready to learn about adding plugins to your URDF so that different aspects of your robot and the simulated environment can be controlled. See ROS Motor and Sensor Plugins (http://gazebosim.org/tutorials?tut=ros_gzplugins).

(https://www.youtube.com/channel/UCJyq