🡐 Back (/tutorials?cat=install)

Gazebo plugins in ROS

Tutorial: Using Gazebo plugins with ROS

Gazebo plugins give your URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input. In this tutorial we explain both how to setup preexisting plugins and how to create your own custom plugins that can work with ROS.

Prerequisites

Make sure you have the RRBot setup as described in the previous tutorial on URDFs (http://gazebosim.org/tutorials/?tut=ros_urdf). Also make sure you have understood the use of the `<gazebo>` element within the URDF description, from that same tutorial.

Plugin Types

Gazebo supports several plugin types (http://gazebosim.org/tutorials?tut=plugins_hello_world&cat=write_plugin), and all of them can be connected to ROS, but only a few types can be referenced through a URDF file:

1. ModelPlugins (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1ModelPlugin.html), to provide access to the physics::Model (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1physics_1_1Model.html) API
2. SensorPlugins (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1SensorPlugin.html), to provide access to the sensors::Sensor (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1sensors_1_1Sensor.html) API
3. VisualPlugins (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1VisualPlugin.html), to provide access to the rendering::Visual (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1rendering_1_1Visual.html) API

Adding a `ModelPlugin`

In short, the `ModelPlugin` is inserted in the URDF inside the `<robot>` element. It is wrapped with the `<gazebo>` pill, to indicate information passed to Gazebo. For example:

```
<robot>
  ... robot description ...
  <gazebo>
    <plugin name="differential_drive_controller" filename="libdiffdrive_plugin.so">
      ... plugin parameters ...
    </plugin>
  </gazebo>
  ... robot description ...
</robot>
```

Upon loading the robot model (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1physics_1_1Model.html) within Gazebo, the `diffdrive_plugin` code will be given a reference to the model itself, allowing it to manipulate it. Also, it will be give a reference to the SDF element (http://osrf-distributions.s3.amazonaws.com/sdformat/api/dev/classsdf_1_1Element.html) of itself, in order to read the plugin parameters passed to it.

Adding a `SensorPlugin`

Specifying sensor plugins is slightly different. Sensors (http://gazebosim.org/api/dev/group__gazebo__sensors.html) in Gazebo are meant to be attached to links, so the `<gazebo>` element describing that sensor must be given a reference to that link. For example:

```
<robot>
  ... robot description ...
  <link name="sensor_link">
    ... link description ...
  </link>

  <gazebo reference="sensor_link">
    <sensor type="camera" name="camera1">
      ... sensor parameters ...
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        ... plugin parameters ..
      </plugin>
    </sensor>
  </gazebo>

</robot>
```

Upon loading the robot model within Gazebo, the `camera_controller` code will be given a reference to the sensor, providing access to its API. Also, it will be give a reference to the SDF element of itself, in order to read the plugin parameters passed to it.

Plugins available in gazebo_plugins

The following sections document all of the plugins available in the `gazebo_plugins`. We suggest you review them in order because more detail is covered in the first couple of plugins and you can learn some of the concepts from the various plugins' documentation.

The names of each section is derived from the plugin class name. For example, "Block Laser" is from the **`GazeboRosBlockLaser`** class and can be found in the file `gazebo_plugins/src/gazebo_ros_block_laser.cpp`.

If there are some sections blank, it means that this author got tired of documenting every plugin and you should fill in the area with your experience should you have knowledge and examples of how to use the particular plugin.

Camera

Description: provides ROS interface for simulating cameras such as wge100*camera by publishing the CameraInfo and Image ROS messages as described in sensor*msgs.

RRBot Example

In this section, we will review a simple RGB camera attached to the end of the RRBot pendulum arm. You can look inside `rrbot.xacro` to follow the explanation. The first elements of this block are an extra link and joint added to the URDF file that represents the camera. We are just using a simple red box to represent the camera, though typically you could use a mesh file for a better representation.

```
<joint name="camera_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz="${camera_link} 0 ${height3 - axel_offset*2}" rpy="0 0 0"/>
  <parent link="link3"/>
  <child link="camera_link"/>
</joint>

<!-- Camera -->
<link name="camera_link">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
  <box size="${camera_link} ${camera_link} ${camera_link}"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
  <box size="${camera_link} ${camera_link} ${camera_link}"/>
    </geometry>
    <material name="red"/>
  </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>
```

A Xacro property is also defined:

```
<xacro:property name="camera_link" value="0.05" /> <!-- Size of square 'camera' box -->
```

You should be able to launch the RRBot and see a red box attached to the end of the arm.

Next we will review the Gazebo plugin that gives us the camera functionality and publishes the image to a ROS message. In the RRBot we have been following the convention of putting Gazebo elements in the `rrbot.gazebo` file:

```
  <!-- camera -->
  <gazebo reference="camera_link">
    <sensor type="camera" name="camera1">
      <update_rate>30.0</update_rate>
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>800</width>
          <height>800</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
        <noise>
          <type>gaussian</type>
          <!-- Noise is sampled independently per pixel on each frame.
               That pixel's noise value is added to each of its color
               channels, which at that point lie in the range [0,1]. -->
          <mean>0.0</mean>
          <stddev>0.007</stddev>
        </noise>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>rrbot/camera1</cameraName>
        <imageTopicName>image_raw</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera_link</frameName>
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
      </plugin>
    </sensor>
  </gazebo>
```

Let's discuss some of the properties of this plugin...

```
  <gazebo reference="camera_link">
```

The link name "camera_link" must match the name of the link we added to the Xacro URDF.

```
    <sensor type="camera" name="camera1">
```

The sensor name "camera1" must be unique from all other sensor names. The name is not used many places except for within Gazebo plugins you can access

```
      <update_rate>30.0</update_rate>
```

Number of times per second a new camera image is taken within Gazebo. This is the maximum update rate the sensor will attempt during simulation but it could fall behind this target rate if the physics simulation runs faster than the sensor generation can keep up.

```
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>800</width>
          <height>800</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
```

Fill in these values to match the manufacturer's specs on your physical camera hardware. One thing to note is that the pixels are assumed to be square.

Additionally, the near and far clips are simulation-specific parameters that give an upper and lower bound to the distance in which the cameras can see objects in the simulation. This is specified in the camera's optometry frame.

```
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
```

This is where the actual `gazebo_ros/gazebo_ros_camera.cpp` file is linked to, as a shared object.

```
        <cameraName>rrbot/camera1</cameraName>
        <imageTopicName>image_raw</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
```

Here we define the rostopic the camera will be publishing to, for both the image topic and the camera info topic. For RRBot, you should subscribe to:

```
/rrbot/camera1/image_raw
/rrbot/camera1/camera_info
```

```
        <frameName>camera_link</frameName>
```

The coordinate frame the image is published under in the tf tree.

Running the RRBot Example

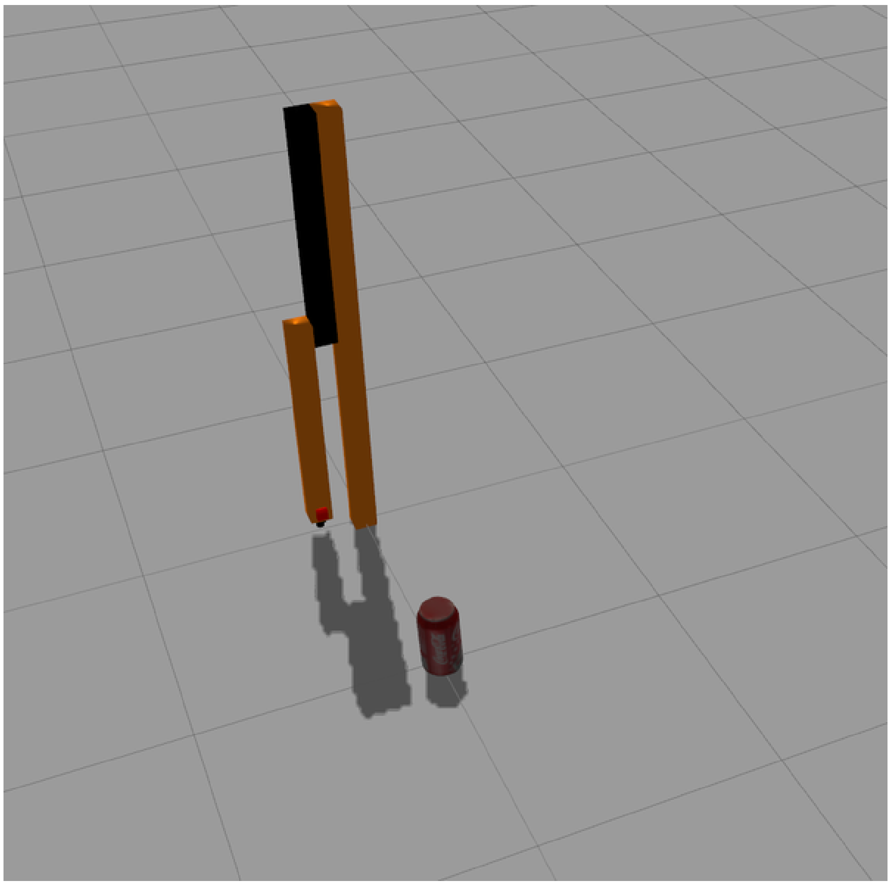
After you have saved both `rrbot.xacro` and `rrbot.gazebo` , you should be able to launch both Rviz and Gazebo in separate terminals:

```
roslaunch rrbot_gazebo rrbot_world.launch
roslaunch rrbot_description rrbot_rviz.launch
```

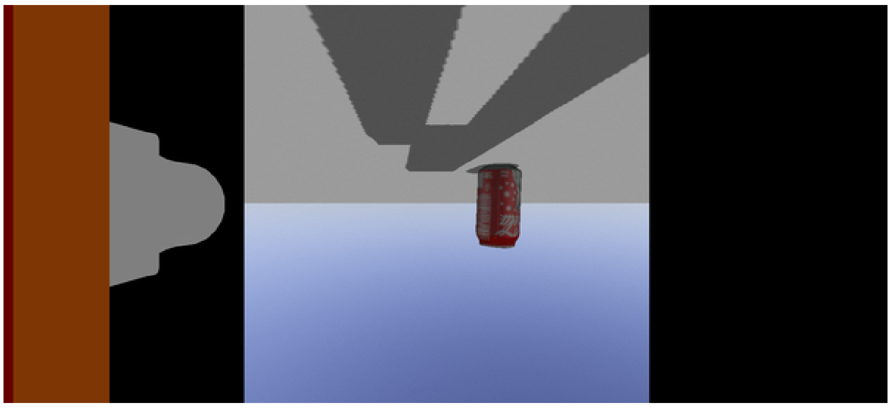
In Rviz, add a "Camera" display and under "Image Topic" set it to `/rrbot/camera1/image_raw` .

You should see a camera view of your Gazebo environment. In the following two pictures, a soda can was added to the environment for better visuals.

The coke can added:



The corresponding camera view after the pendulum has fallen:



Multicamera

Description: synchronizes multiple camera's shutters such that they publish their images together. Typically used for stereo cameras, uses a very similar interface as the plain `Camera` plugin

Note: currently only supports stereo cameras. See Github issue (https://github.com/osrf/gazebo_ros_pkgs/issues/13).

Atlas Code Example

In this code example there is both a left and right camera:

```
<gazebo reference="left_camera_frame">
  <sensor type="multicamera" name="stereo_camera">
    <update_rate>30.0</update_rate>
    <camera name="left">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
    <camera name="right">
      <pose>0 -0.07 0 0 0 0</pose>
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
    <plugin name="stereo_camera_controller" filename="libgazebo_ros_multicamera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>multisense_sl/camera</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>left_camera_optical_frame</frameName>
      <!--<rightFrameName>right_camera_optical_frame</rightFrameName>-->
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

Depth Camera

Description: simulates a sensor like a Kinect, which is duplicated in the Kinect plugin. Will probably be merged in the future.

Openni Kinect

'''Description:''' Simulates an Xbox-Kinect, publishes the same topics as the corresponding ROS drivers for the Xbox kinect as documented in the Fuerte documentation here (http://www.ros.org/wiki/openni_camera).

```
<gazebo>
  <plugin name="${link_name}_controller" filename="libgazebo_ros_openni_kinect.so">
    <baseline>0.2</baseline>
    <alwaysOn>true</alwaysOn>
    <updateRate>1.0</updateRate>
    <cameraName>${camera_name}_ir</cameraName>
    <imageTopicName>/${camera_name}/depth/image_raw</imageTopicName>
    <cameraInfoTopicName>/${camera_name}/depth/camera_info</cameraInfoTopicName>
    <depthImageTopicName>/${camera_name}/depth/image_raw</depthImageTopicName>
    <depthImageInfoTopicName>/${camera_name}/depth/camera_info</depthImageInfoTopicName>
    <pointCloudTopicName>/${camera_name}/depth/points</pointCloudTopicName>
    <frameName>${frame_name}</frameName>
    <pointCloudCutoff>0.5</pointCloudCutoff>
    <distortionK1>0.00000001</distortionK1>
    <distortionK2>0.00000001</distortionK2>
    <distortionK3>0.00000001</distortionK3>
    <distortionT1>0.00000001</distortionT1>
    <distortionT2>0.00000001</distortionT2>
    <CxPrime>0</CxPrime>
    <Cx>0</Cx>
    <Cy>0</Cy>
    <focalLength>0</focalLength>
    <hackBaseline>0</hackBaseline>
  </plugin>
</gazebo>
```

GPU Laser

Description: simulates laser range sensor by broadcasting LaserScan message as described in sensor_msgs. See Hokuyo Laser Scanners Reference (http://ros.org/wiki/hokuyo_node).

RRBot Example

See the RRBot Example for adding a Camera to RRBot before reviewing this example. Similar to adding a camera, we will add a new link and joint to the Xacro URDF of the RRBot. This time, instead of using just a rectangle for the visual model, we'll use a mesh:

```
  <joint name="hokuyo_joint" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0 0 ${height3 - axel_offset/2}" rpy="0 0 0"/>
    <parent link="link3"/>
    <child link="hokuyo_link"/>
  </joint>

  <!-- Hokuyo Laser -->
  <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://rrbot_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>
```

Now we'll add the plugin information to `rrbot.gazebo` , again as we did for the camera example:

```
  <!-- hokuyo -->
  <gazebo reference="hokuyo_link">
    <sensor type="gpu_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_gpu_laser.so">
        <topicName>/rrbot/laser/scan</topicName>
        <frameName>hokuyo_link</frameName>
      </plugin>
    </sensor>
  </gazebo>
```

Most of the properties are self-explanatory, but we'll review some below:

```
      <visualize>false</visualize>
```

When true, a semi-translucent laser ray is visualized within the scanning zone of the gpu laser. This can be an informative visualization, or an nuisance.

More documentation on the `<sensor>` and `<ray>` elements can be found in the SDF Documentation (http://gazebosim.org/sdf/dev.html#sensor225).

```
<topicName>/rrbot/laser/scan</topicName>
<frameName>hokuyo_link</frameName>
```

Set these to the ROS topic name you would like to publish the laser scans to, and the transform frame you would like TF to use.

Running the RRBot Example

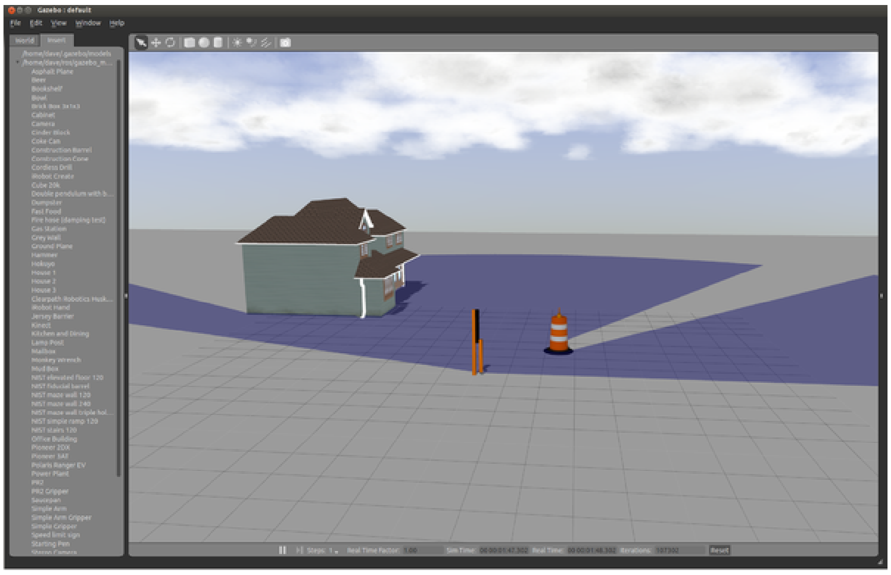
After you have saved both `rrbot.xacro` and `rrbot.gazebo`, you should be able to launch both Rviz and Gazebo in separate terminals:

```
roslaunch rrbot_gazebo rrbot.launch
roslaunch rrbot_description rrbot_rviz.launch
```

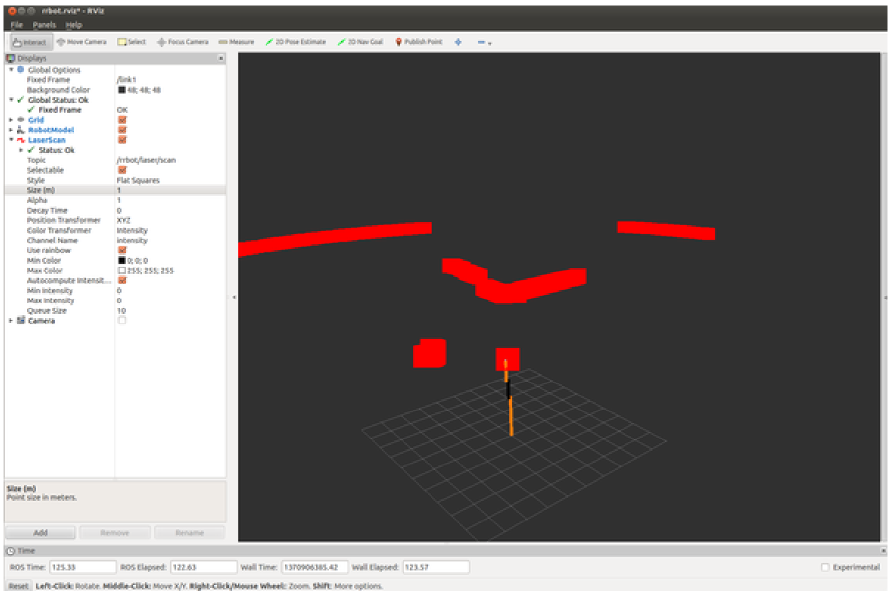
In Rviz, add a "LaserScan" display and under "Topic" set it to `/rrbot/camera1/image_raw`.

You should see a faint laser scan line in your Gazebo environment. While the pendulum is swinging, you should also see the laser scan swing. If the scan is too faint, you can up the size of the laser scan in the properties of the LaserScan display in Rviz. A size of 1m is very easy to see. In the following two pictures, a house and construction barrel was added to the environment for better visuals.

View from Gazebo:



The corresponding laser view from Rviz:



Laser

Description: the non-GPU version of `GPU Laser`, but essentially uses the same code. See GPU Laser for documentation.

To run with RRBot, open `rrbot.gazebo` and change the following two lines.

replace

```
    <sensor type="gpu_ray" name="head_hokuyo_sensor">
```

with

```
    <sensor type="ray" name="head_hokuyo_sensor">
```

and replace

```
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_gpu_laser.so">
```

with

```
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
```

save, then launch the same launch files as for GPU Laser.

Block Laser

Description: provides grid style laser range scanner simulation (e.g. Velodyne).

F3D (Force Feedback Ground Truth)

Description: broadcasts external forces on a body in simulation over WrenchStamped message as described in geometry_msgs.

Force

Description: ROS interface for applying Wrench (geometry_msgs) on a body in simulation.

IMU (GazeboRosImu)

Description: simulates IMU sensor. Measurements are computed by the ROS plugin, not by Gazebo. See usage snippet sample below for implementation.

```
<robot>
:
  <gazebo>
    <plugin name="imu_plugin" filename="libgazebo_ros_imu.so">
      <alwaysOn>true</alwaysOn>
      <bodyName>base_footprint</bodyName>
      <topicName>imu</topicName>
      <serviceName>imu_service</serviceName>
      <gaussianNoise>0.0</gaussianNoise>
      <updateRate>20.0</updateRate>
    </plugin>
  </gazebo>
</robot>
```

IMU sensor (GazeboRosImuSensor)

Description: simulates an Inertial Motion Unit sensor, the main differences from **IMU** (GazeboRosIMU) are: - inheritance from SensorPlugin instead of ModelPlugin, - measurements are given by gazebo ImuSensor instead of being computed by the ros plugin, - gravity is included in inertial measurements.

```
  <gazebo reference="imu_link">
    <gravity>true</gravity>
    <sensor name="imu_sensor" type="imu">
      <always_on>true</always_on>
      <update_rate>100</update_rate>
      <visualize>true</visualize>
      <topic>__default_topic__</topic>
      <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
        <topicName>imu</topicName>
        <bodyName>imu_link</bodyName>
        <updateRateHZ>10.0</updateRateHZ>
        <gaussianNoise>0.0</gaussianNoise>
        <xyzOffset>0 0 0</xyzOffset>
        <rpyOffset>0 0 0</rpyOffset>
        <frameName>imu_link</frameName>
      </plugin>
      <pose>0 0 0 0 0 0</pose>
    </sensor>
  </gazebo>
```

Joint Pose Trajectory

Description: listens to a joint*trajectory*action and plays back the set of joint positions. Sets the set of joints to exact positions without regards to simulated physics and forces.

P3D (3D Position Interface for Ground Truth)

Description: broadcasts the inertial pose of any body in simulation via Odometry message as described in nav_msgs via ROS topic.

Projector

Description: projects a static texture from a source outwards, such as used with the PR2's original head camera sensor. See API documentation (http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1rendering_1_1Projector.html) for more information.

Prosilica Camera

Description: Simulates interfaces exposed by a ROS Prosilica Camera (http://www.ros.org/wiki/prosilica_camera). Here's an example URDF Xacro macro (https://bitbucket.org/hsu/nasa_r2_simulator/src/5ee1de067038749dcc133ed7cf87b45715cc4457/r2_gazebo/urdf/sensors/grasshopper2.gazebo.xacro?at=hsu).

Bumper

Description: provides contact feedback via ContactsState message (http://docs.ros.org/api/gazebo_msgs/html/msg/ContactsState.html).

```
<gazebo>
  <plugin name="${name}_gazebo_ros_bumper_controller" filename="libgazebo_ros_bumper.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>${update_rate}</updateRate>
    <bumperTopicName>${name}_bumper</bumperTopicName>
    <frameName>world</frameName>
  </plugin>
</gazebo>
```

Differential Drive

Description model plugin that provides a basic controller for differential drive robots in Gazebo. You need a well defined differential drive robot to use this plugin.

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>${update_rate}</updateRate>
    <leftJoint>base_link_right_wheel_joint</leftJoint>
    <rightJoint>base_link_left_wheel_joint</rightJoint>
    <wheelSeparation>0.5380</wheelSeparation>
    <wheelDiameter>0.2410</wheelDiameter>
    <torque>20</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>base_footprint</robotBaseFrame>
  </plugin>
</gazebo>
```

Skid Steering Drive

Description model plugin that provides a basic controller for skid steering drive robots in Gazebo (Pioneer 3AT for instance).

```
<gazebo>
  <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>front_left_wheel_joint</leftFrontJoint>
    <rightFrontJoint>front_right_wheel_joint</rightFrontJoint>
    <leftRearJoint>back_left_wheel_joint</leftRearJoint>
    <rightRearJoint>back_right_wheel_joint</rightRearJoint>
    <wheelSeparation>0.4</wheelSeparation>
    <wheelDiameter>0.215</wheelDiameter>
    <robotBaseFrame>base_link</robotBaseFrame>
    <torque>20</torque>
    <topicName>cmd_vel</topicName>
    <broadcastTF>false</broadcastTF>
  </plugin>
</gazebo>
```

Video Plugin

Description visual plugin that displays a ROS image stream on an OGRE Texture inside gazebo. This plugin does not modify the texture of one of the existing link surfaces, but creates a new texture on top of it. The texture will be created on the XY plane, visible from the +Z side. The plugin requires a pixel size while constructing the texture, and will resize incoming ROS image messages to match if they are a different size.

```
<gazebo reference="display_screen_link">
  <visual>
    <plugin name="display_video_controller" filename="libgazebo_ros_video.so">
      <topicName>image</topicName>
      <height>120</height>
      <width>160</width>
    </plugin>
  </visual>
</gazebo>
```

Planar Move Plugin

Description model plugin that allows arbitrary objects (for instance cubes, spheres and cylinders) to be moved along a horizontal plane using a geometry_msgs/Twist message. The plugin works by imparting a linear velocity (XY) and an angular velocity (Z) to the object every cycle.

Here is a full URDF example that demonstrates how to control a floating box inside gazebo using this plugin, using different visual and collision elements. Note: The object needs to have sufficient inertia to prevent undesirable motions - which can occur as a reaction to the supplied velocity. You can try increasing inertia until the object moves as desired. It is also good to have the center of mass close to the ground.

```
<robot name="test_model">

  <!-- root link, on the ground just below the model origin -->
  <link name="base_footprint">
   <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <box size="0.001 0.001 0.001" />
      </geometry>
    </visual>
  </link>

  <joint name="base_link_joint" type="fixed">
    <origin xyz="0.0 0 1.25" rpy="0 0 0" />
    <parent link="base_footprint"/>
    <child link="base_link" />
  </joint>

  <!-- the model -->
  <link name="base_link">
    <inertial>
      <mass value="50" />
      <origin xyz="0 0 -1.25" />
      <inertia ixx="50.0" ixy="0.0" ixz="0.0"
        iyy="50.0" iyz="0.0"
        izz="50.0" />
    </inertial>
    <visual>
      <geometry>
        <box size="0.5 0.5 1.0" /> <!-- does not need to match collision -->
      </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 -1.0" />
      <geometry>
        <cylinder length="0.5" radius="0.25" />
      </geometry>
    </collision>
  </link>

  <gazebo>
    <plugin name="object_controller" filename="libgazebo_ros_planar_move.so">
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <odometryRate>20.0</odometryRate>
      <robotBaseFrame>base_footprint</robotBaseFrame>
    </plugin>
  </gazebo>

</robot>
```

Template

Description: an example c++ plugin template for anyone who wants to write their own plugin.

Issue report, contribution

Gazebo-ROS plugins are stored in a ROS package. See gazebo_plugins wiki page (http://wiki.ros.org/gazebo_plugins) about how you can contribute.

3rd party plugins

In addition to the plugins explained above, there are also a number of 3rd party Gazebo-ROS plugins. Some of them are found on ros.org (example of search keyword (https://www.google.com/search?client=ubuntu&channel=fs&q=site%3Aros.org+*_gazebo_plugins&ie=utf-8&oe=utf-8)). If a 3rd party plugin is useful and generic enough, please consider pulling it into the official gazebo_plugins package (wiki page) (http://wiki.ros.org/gazebo_plugins) by opening a suggestion at the issue tracker of each repository.

Next Steps

Next we will analyze the `ros_control` packages integrated with Gazebo for tight controller/actuator/simulator integration Actuators, controllers, and ros_control (http://gazebosim.org/tutorials/?tut=ros_control).