3D Robot Modeling and Simulation in ROS

Nicola Castaman
nicola.castaman@dei.unipd.it

What we are going to learn

- Create a 3D model of a robot
- Provide movements, physical limits, inertia, and other physical aspects to our robot
- Add simulated sensors to our 3D model
- Use the model on the simulator

Robot Modeling Using URDF

The **Unified Robot Description Format (URDF)** is an XML format that allow to represent the kinematic and dynamic description of the robot, visual representation of the robot, and the collision model of the robot.

The commonly used URDF tags to compose a URDF robot model are:

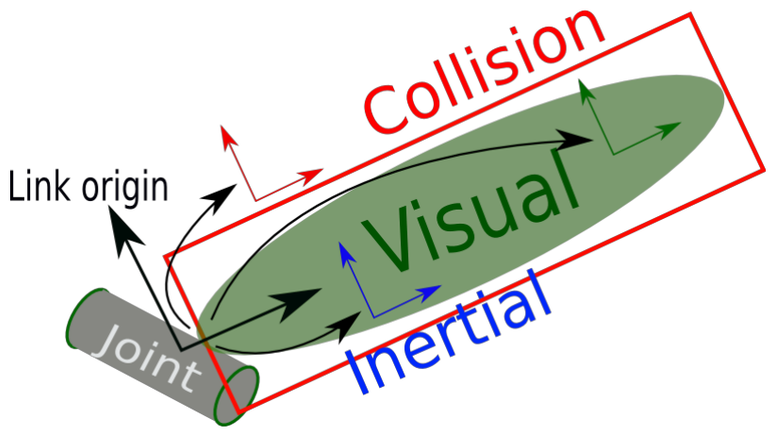
- `<robot>`
- `<link>`
- `<joint>`
- `<gazebo>`

Robot Modeling Using URDF

Link

The `<link>` tag represents a single link of a robot. We can model a robot link and its properties including size, shape, color, and import a 3D mesh to represent the robot link. We can also provide dynamic properties of the link such as inertial matrix and collision properties.

```
<link name="[name of the link]">
  <visual> … </visual>
  <collision> … </collision>
  <inertial> … </inertial>
</link>
```



`<visual>` represents the real link of the robot, and the area surrounding the real link is the `<collision>` section.

`<collision>` encapsulates the real link to detect collision before hitting the real link

`<inertial>` define the inertia of the link

Robot Modeling Using URDF

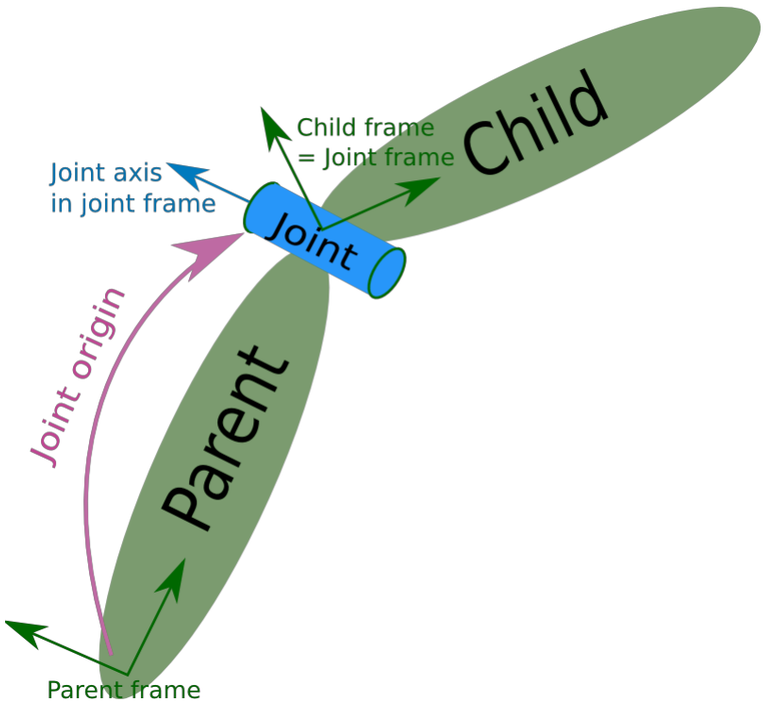
Joint

The `<joint>` tag represents a robot joint. We can specify the kinematics and dynamics of the joint and also set the limits of the joint movement and its velocity. The joint tag supports the different types of joints such as **revolute**, **continuous**, **prismatic**, **fixed**, **floating**, and **planar**.

```
<joint name="[name of the joint]" type="[type of the tag]">
  <parent link="parent_link"/>
  <child link="child_link"/>

  <axis … />

  <calibration … />
  <dynamics damping … />
  <limit effort … />
</joint>
```



`<parent>` represents parent link

`<child>` represents parent link

`<axis>` define the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints

`<limit>` define lower and upper joint limit

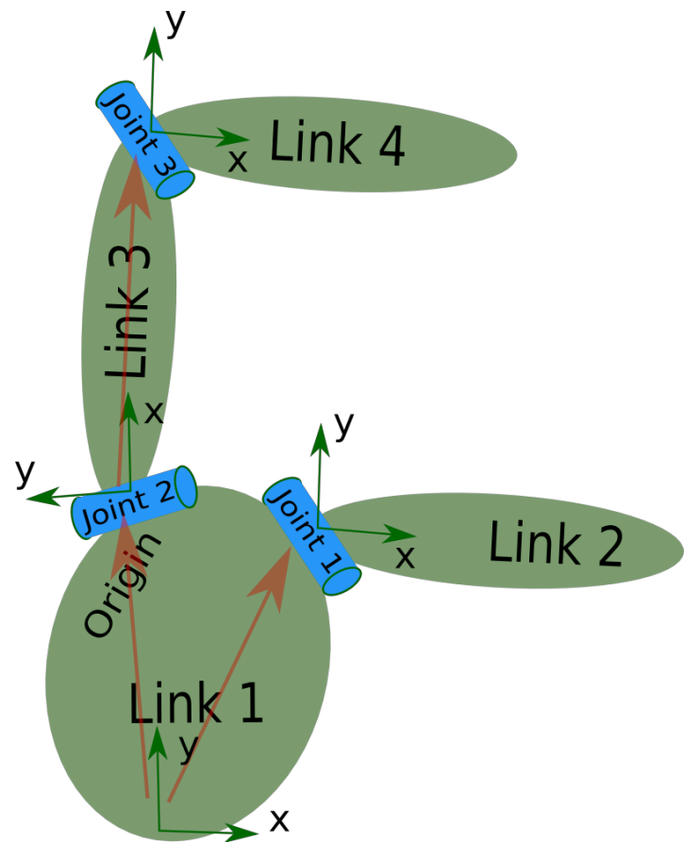
Robot Modeling Using URDF

Robot

The `<robot>` tag encapsulates the entire robot model that can be represented using URDF.

```
<robot name="[name of the robot]”>
  <link> … </link>
  <link> … </link>

  <joint> … </joint>
  <joint> … </joint>
</robot>
```



Gazebo

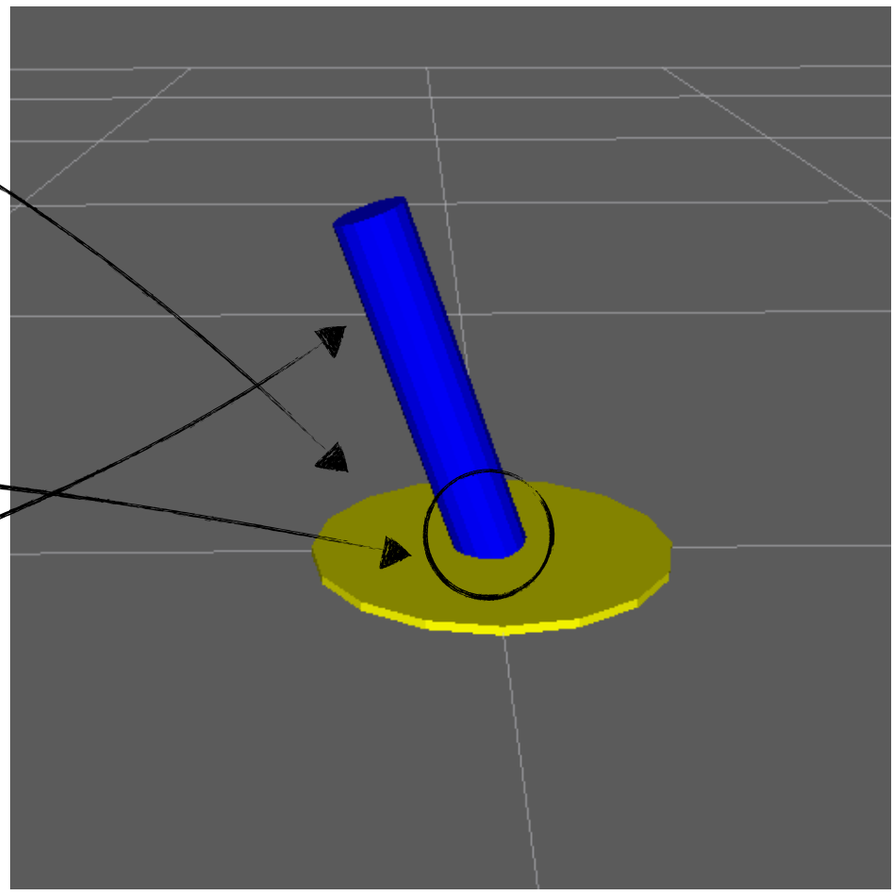
The `<gazebo>` tag is used when we include the simulation parameters of the Gazebo simulator inside URDF.

```
<gazebo reference="link_1">
  <material>Gazebo/Black</material>
</gazebo>
```

Robot Modeling Using URDF

Creating our first URDF

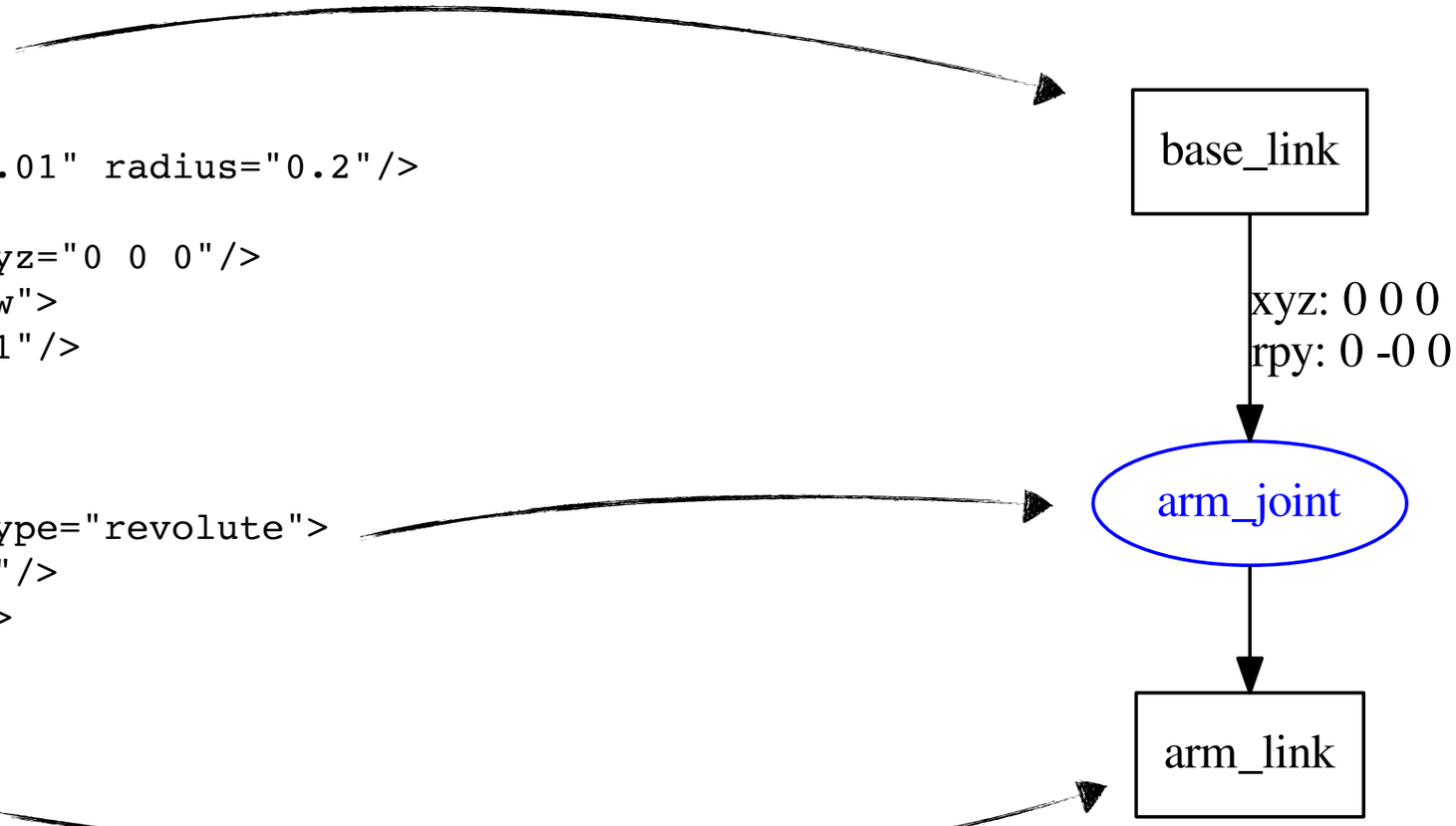
```
<?xml version="1.0"?>
  <robot name=“simple_model”>
    <link name=“base_link">
      <visual>
        <geometry>
          <cylinder length="0.01" radius="0.2"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="yellow">
          <color rgba="1 1 0 1"/>
        </material>
      </visual>
    </link>
    <joint name="arm_joint" type="revolute">
      <parent link="base_link"/>
      <child link="pan_link"/>
      <origin xyz="0 0 0”/>
      <axis xyz="1 0 0" />
    </joint>
    <link name="arm_link">
      <visual>
        <geometry>
          <cylinder length="0.4" radius="0.04"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0.2”/>
        <material name="blue">
          <color rgba="0 0 1 1"/>
        </material>
      </visual>
    </link>
</robot>
```



Robot Modeling Using URDF

Creating our first URDF

```
<?xml version="1.0"?>
  <robot name=“simple_model”>
    <link name=“base_link">
      <visual>
        <geometry>
          <cylinder length="0.01" radius="0.2"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="yellow">
          <color rgba="1 1 0 1"/>
        </material>
      </visual>
    </link>
    <joint name="arm_joint" type="revolute">
      <parent link="base_link"/>
      <child link="pan_link"/>
      <origin xyz="0 0 0”/>
      <axis xyz="1 0 0" />
    </joint>
    <link name="arm_link">
      <visual>
        <geometry>
          <cylinder length="0.4" radius="0.04"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0.2”/>
        <material name="blue">
          <color rgba="0 0 1 1"/>
        </material>
      </visual>
    </link>
</robot>
```



Robot Modeling Using URDF

Adding collision and physical properties

```
<link>
  …
  <collision>
    <geometry>
      <cylinder length="0.4" radius="0.04"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0.2"/>
  </collision>
  <inertial>
    <mass value="1"/>
    <inertia ixx="0.5" ixy="0.0" ixz=“0.0”
             iyy="0.5" iyz="0.0" izz="0.5"/>
  </inertial>
</link>
```

The `<collision>` and `<inertia>` parameters are required in each link; otherwise, Gazebo will not load the robot model properly.

Robot Modeling Using URDF

Meshes

Instead of a primitive shape, we can insert a mesh file to a link using the `<mesh>` tag.

```
<link>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0" />
    <geometry>
      <mesh filename=“package://model_tutorial/meshes/mesh.dae” />
    </geometry>
  </visual>
</link>
```

The meshes can be imported in a number of different formats. STL and DAE are fairly common, but DAE can have its own color data, meaning you don’t have to specify the color/material

Robot Modeling Using Xacro

What URDF is missing

- Simplicity → Complex robot models generate extremely long files
- Reusability → It is not possible reuse a URDF block multiple times
- Modularity → The URDF is single file and can’t include other URDF files inside it
- Programmability → No variable, constants, mathematical expressions, conditional statement, and so on

The robot modeling using xacro meets all these conditions!

To use xacro, we need to specify a namespace so that the file is parsed properly.

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name=“[robot name]”>
```

The file extension will be `.xacro` instead of `.urdf`.

Robot Modeling Using Xacro

Properties

We can declare constants or properties that are the named values inside the xacro file, which can be used anywhere in the code.

We can keep constants like values of links and joints and it will be easier to change these values rather than finding the hard coded values and replacing them.

```
<xacro:property name="base_link_length" value="0.01" />
<xacro:property name="base_link_radius" value="0.2" />
<xacro:property name="arm_link_length" value="0.4" />
<xacro:property name=“arm_link_radius" value="0.04" />
```

```
<cylinder length="${arm_link_length}" radius="${arm_link_radius}"/>
```

constant inside `${...}`

Math Expression

We can build mathematical expressions inside `${...}` using the basic operations such as `+` , `-`, `*` , `/` , unary minus, and parenthesis.

```
<link name=“arm_link">
  <visual>
    …
    <origin xyz="0 0 ${arm_link_length/2}" rpy="0 0 0"/>
    …
  </visual>
</link>
```

expressions inside `${...}`

Robot Modeling Using Xacro

Macros

We can improve the code readability and reduce the length of complex definition using macros.

Definition


```
<xacro:macro name="inertial_matrix" params="mass">
  <inertial>
    <mass value="${mass}" />
    <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="0.5" />
  </inertial>
</xacro:macro>
```

Use


```
<xacro:inertial_matrix mass="1"/>
```

Visualizing a Robot Model in Rviz

Launch file

```
<launch>

  <param name="robot_description" command="$(find xacro)/xacro --inorder
  '$(find model_tutorial)/urdf/simple_robot.urdf.xacro'" />

  <param name="use_gui" value="true" />

  <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>
```

Joint state publisher is a ROS packages that is used to interact with each joint of the robot. The node inside find the non-fixed joints from the URDF model and publish the joint state values of each joint in the `sensor_msgs/JointState` message format.

If `use_gui` is set to `true`, the joint_state_publisher displays a slider based control window to control each joint.

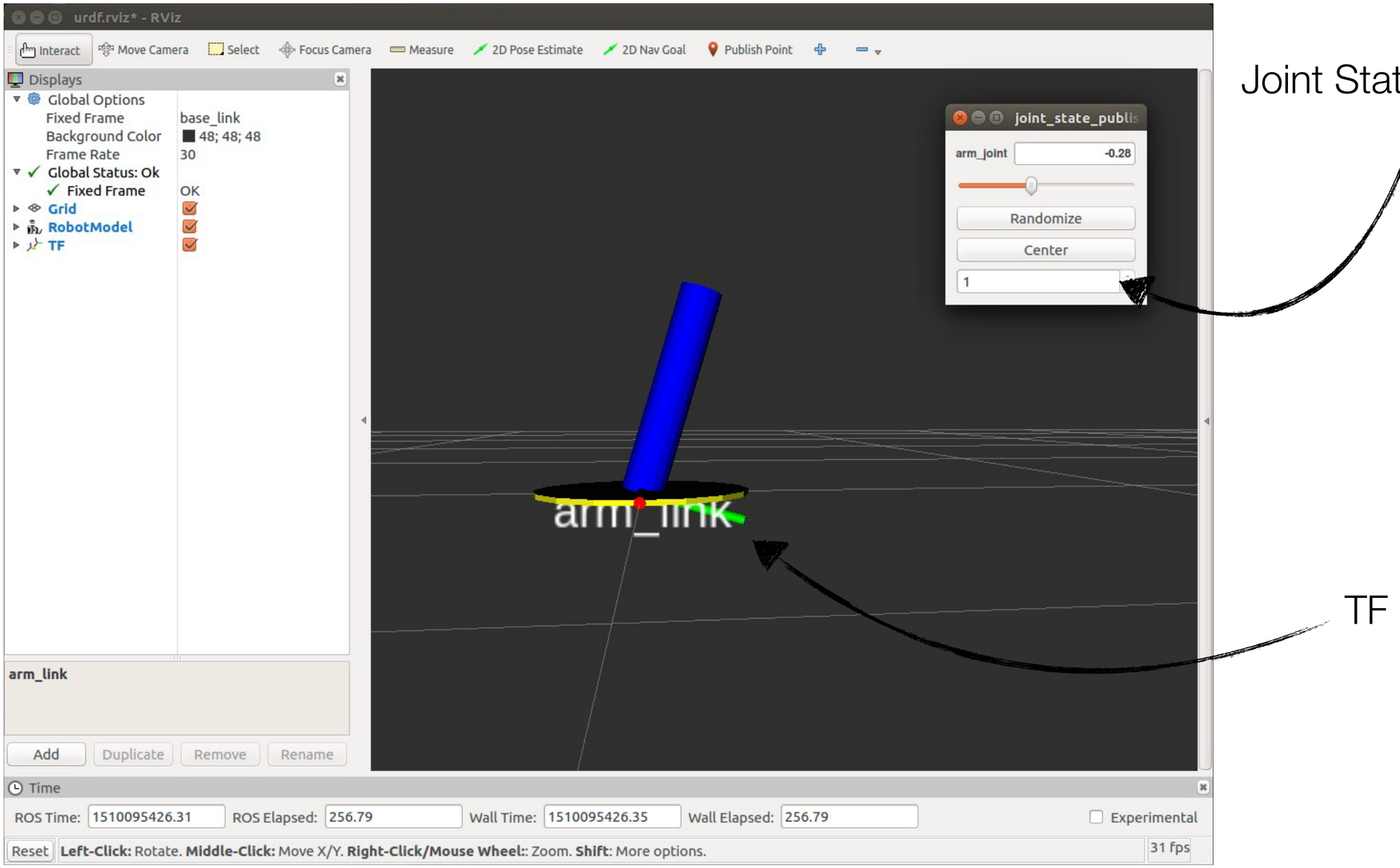
The robot state publisher package helps to publish the state of the robot to tf. This package subscribes to joint states of the robot and publishes the 3D pose of each link using the kinematic representation from the URDF model.

Use

```
$ roslaunch model_tutorial simple_robot.launch
```

Visualizing a Robot Model in Rviz

RViz



Joint State Publisher

TF

Creating a Robotic Manipulator

Arm Specifications

- Degrees of freedom: 3
- Number of links: 4
- Number of joints: 3

Links

Link name	Lenght	Radius	Color
base_link	0.05	0.1	Grey
shoulder_link	0.15	0.06	Red
upper_arm_link	0.55	0.05	White
forearm_link	0.5	45	Red

Joints

Joint name	Type	Angle limits
shoulder_pan_joint	Revolute	-PI to PI
shoulder_lift_joint	Revolute	-PI/2 to PI/2
elbow_joint	Revolute	-3/4PI to 3/4PI

Creating a Robotic Manipulator

Define Arm Properties

```
<!-- Base link properties -->
<property name="base_radius" value="0.1" />
<property name="base_length" value="0.05" />

<!-- Shoulder link properties -->
<property name="shoulder_radius" value="0.06" />
<property name="shoulder_length" value="0.15" />

<!-- Upper Arm link properties -->
<property name="upper_arm_radius" value="0.05" />
<property name="upper_arm_length" value="0.55" />

<!-- Forearm link properties -->
<property name="forearm_radius" value="0.045" />
<property name="forearm_length" value="0.5" />
```

Define Constants

```
<!-- PI -->
<property name="M_PI" value="3.14159"/>
```

Creating a Robotic Manipulator

Define Macros

```
<!-- Cylinder Inertia -->
<xacro:macro name="cylinder_inertia" params="mass radius length">
  <inertia ixx="${1/12*mass*(3*radius*radius+length*length)}" ixy="0.0" ixz="0.0"
           iyy="${1/12*mass*(3*radius*radius+length*length)}" iyz="0.0"
           izz="${1/2*mass*radius*radius}"/>
</xacro:macro>
```

Define Materials

```
<!-- Materials -->
<material name="Grey">
  <color rgba="0.3 0.3 0.3 1.0"/>
</material>

<material name="Red">
  <color rgba="0.8 0.0 0.0 1.0"/>
</material>

<material name="White">
  <color rgba="1.0 1.0 1.0 1.0"/>
</material>
```

Creating a Robotic Manipulator

Create the Base Link

```
<!-- Base Link -->
<link name="base_link">
  <visual>
    <origin xyz="0 0 ${base_length/2}" rpy="0 0 0" />
    <geometry>
      <cylinder length="${base_length}" radius="${base_radius}"/>
    </geometry>
    <material name="Grey" />
  </visual>
  <collision>
    <origin xyz="0 0 ${base_length/2}" rpy="0 0 0" />
    <geometry>
      <cylinder length="${base_length}" radius="${base_radius}"/>
    </geometry>
  </collision>>
  <inertial>
    <mass value="1"/>
    <cylinder_inertia mass="1" radius="${base_radius}" length="${base_length}"/>
  </inertial>
</link>

<gazebo reference="base_link">
  <material>Gazebo/Grey</material>
</gazebo>
```

Visualize the material of the link in Gazebo

Creating a Robotic Manipulator

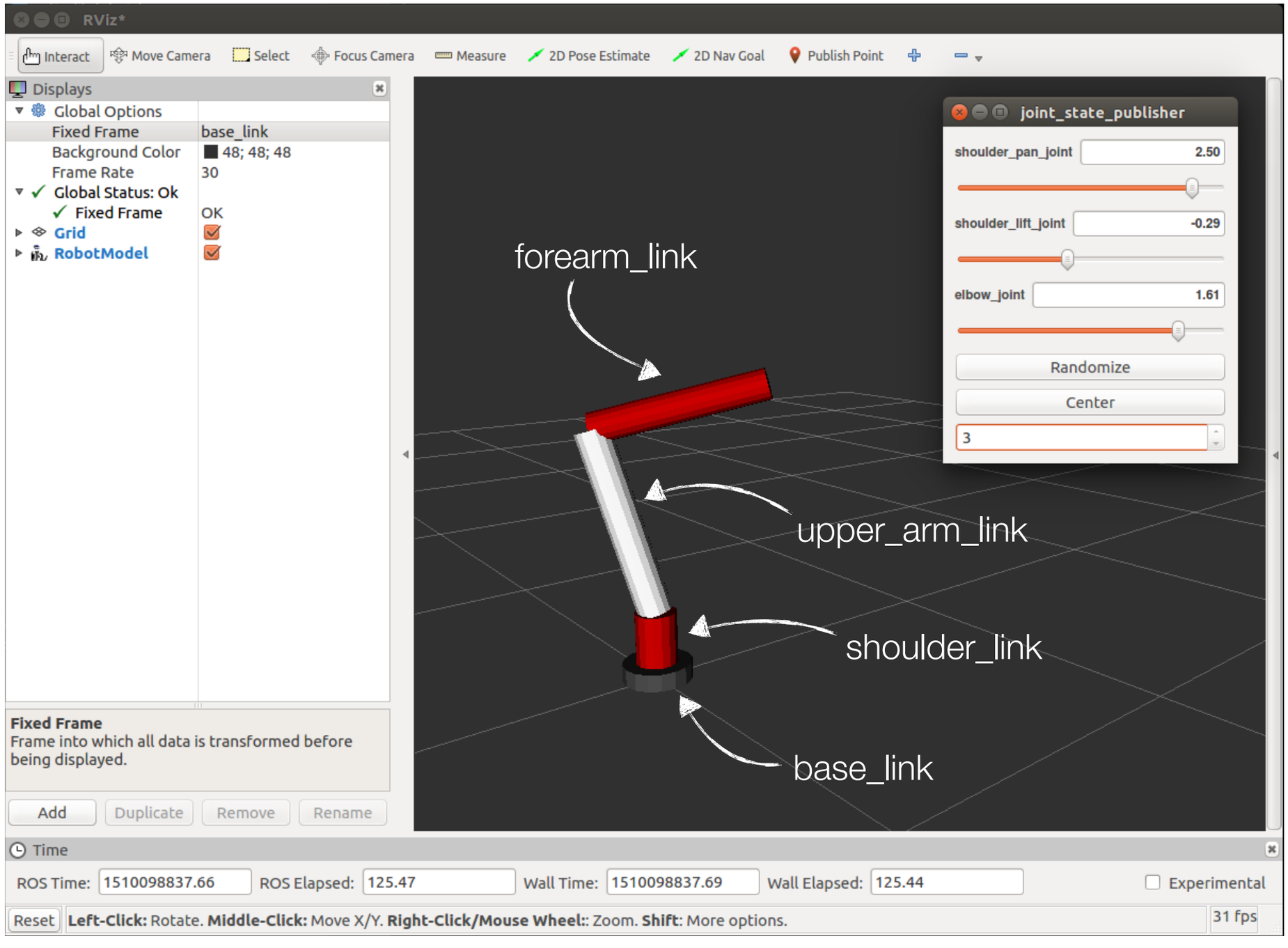
Create the Shoulder Pan Joint

```
<!-- Shoulder Pan Joint -->
<joint name="shoulder_pan_joint" type="revolute">
  <parent link="base_link"/>
  <child link="shoulder_link"/>
  <origin xyz="0 0 ${base_length}" rpy="0 0 0" />
  <axis xyz="0 0 1" />
  <limit effort="300" velocity="1" lower="${-M_PI}" upper="${M_PI}"/>
  <dynamics damping="50" friction="1"/>
</joint>
```

Creating a Robotic Manipulator

The Complete Robotic Manipulator

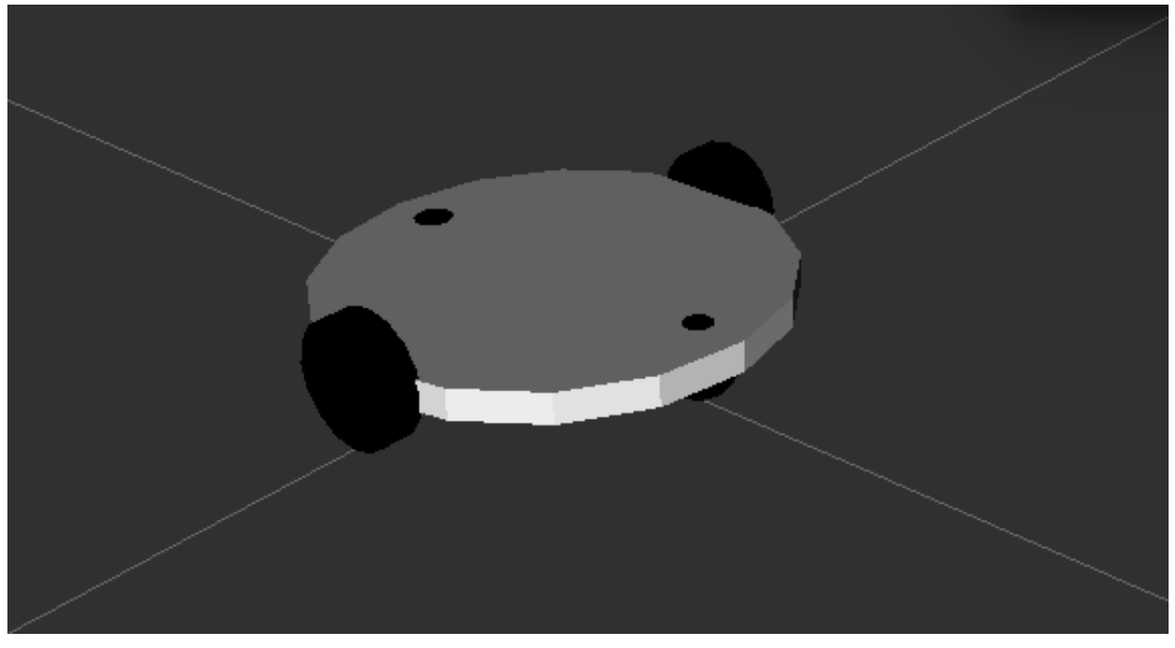
```
$ roslaunch model_tutorial arm.launch
```



Creating a Differential Wheeled Robot

Differential wheeled robot will have two wheels connected on opposite sides of the robot chassis which is supported by one or two caster wheels. The wheels will control the speed of the robot by adjusting individual velocity.

If the two motors are running at the same speed it will move forward or backward. If a wheel is running slower than the other, the robot will turn to the side of the lower speed.



Creating a Differential Wheeled Robot

Creating a Wheel

Defining a robot named “wheel”

```
<?xml version="1.0"?>
<robot name="wheel" xmlns:xacro="http://www.ros.org/wiki/xacro">

  …

  <xacro:macro name="wheel" params="fb lr parent translateX translateY flipY">
    <link name="${fb}_${lr}_wheel">
      …
    </link>

    …

    <joint name="${fb}_${lr}_wheel_joint" type="continuous">
      …
    </joint>

  </xacro:macro>
</robot>
```

Defining a macro named “wheel”

Creating a Differential Wheeled Robot

Include the wheel macro

```
<?xml version="1.0"?>
<robot name="mobile_base" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:include filename="$(find model_tutorial)/urdf/wheel.xacro" />

  …
```

Base footprint

```
<link name="base_footprint">
   <inertial>
     <mass value="0.0001" />
     <origin xyz="0 0 0" />
     <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
         iyy="0.0001" iyz="0.0"
         izz="0.0001" />
   </inertial>
   <visual>
       <origin xyz="0 0 0" rpy="0 0 0" />
       <geometry>
           <box size="0.001 0.001 0.001" />
       </geometry>
   </visual>
 </link>
```

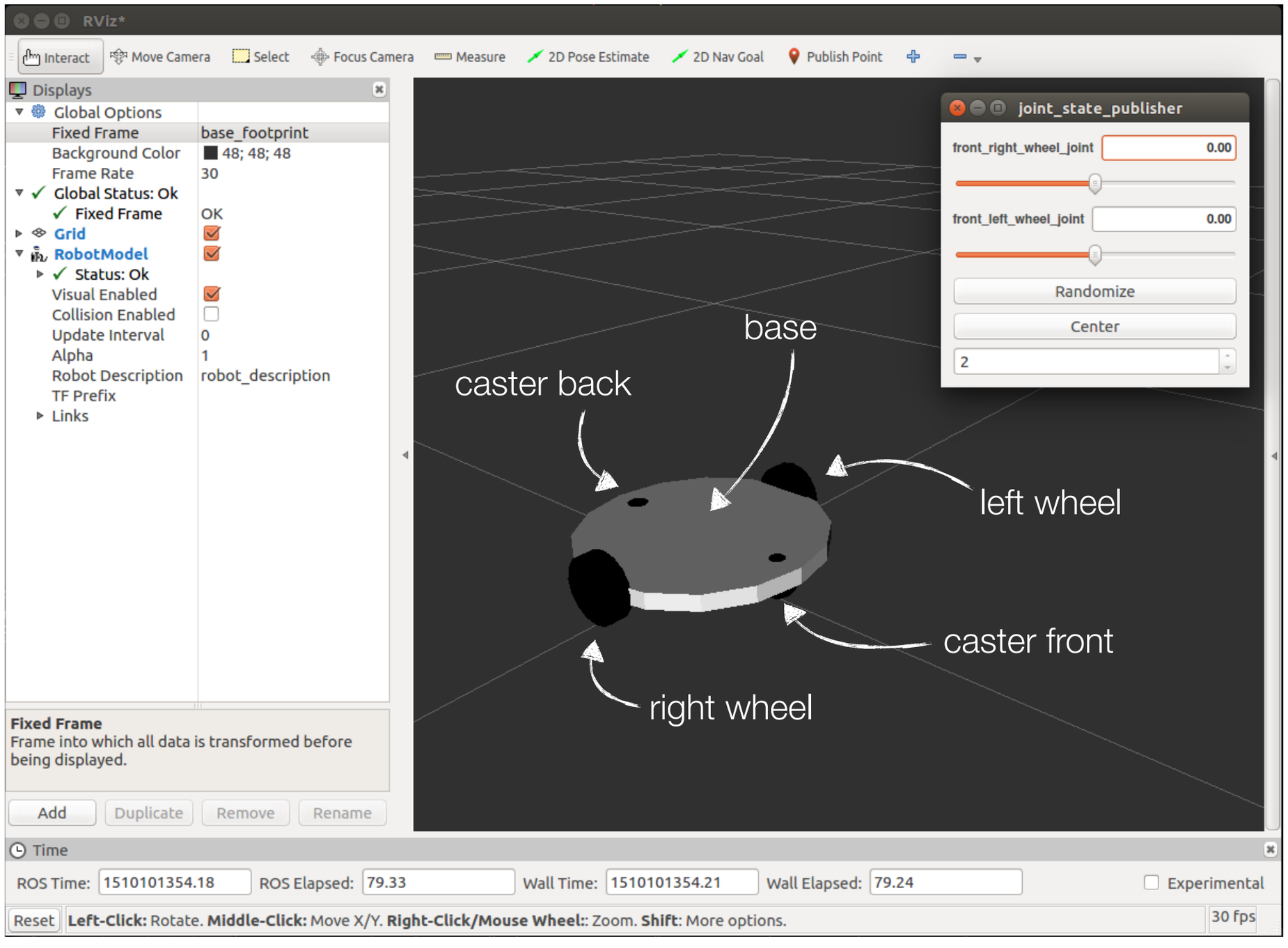
The base_footprint is a fictitious link that is on the ground right below base_link origin

Very small size

Creating a Differential Wheeled Robot

The Complete Mobile Base

```
$ roslaunch model_tutorial mobile_base.launch
```



Simulating the Mobile Robot in Gazebo

Launch File

```
<launch>

  <!-- these are the arguments you can pass this launch file -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(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 -->
  <param name="robot_description" command="$(find xacro)/xacro.py '$(find
model_tutorial)/urdf/mobile_base.xacro'" />

  <!-- Spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
output="screen"
        args="-urdf -model mobile_base -param robot_description"/>

</launch>
```

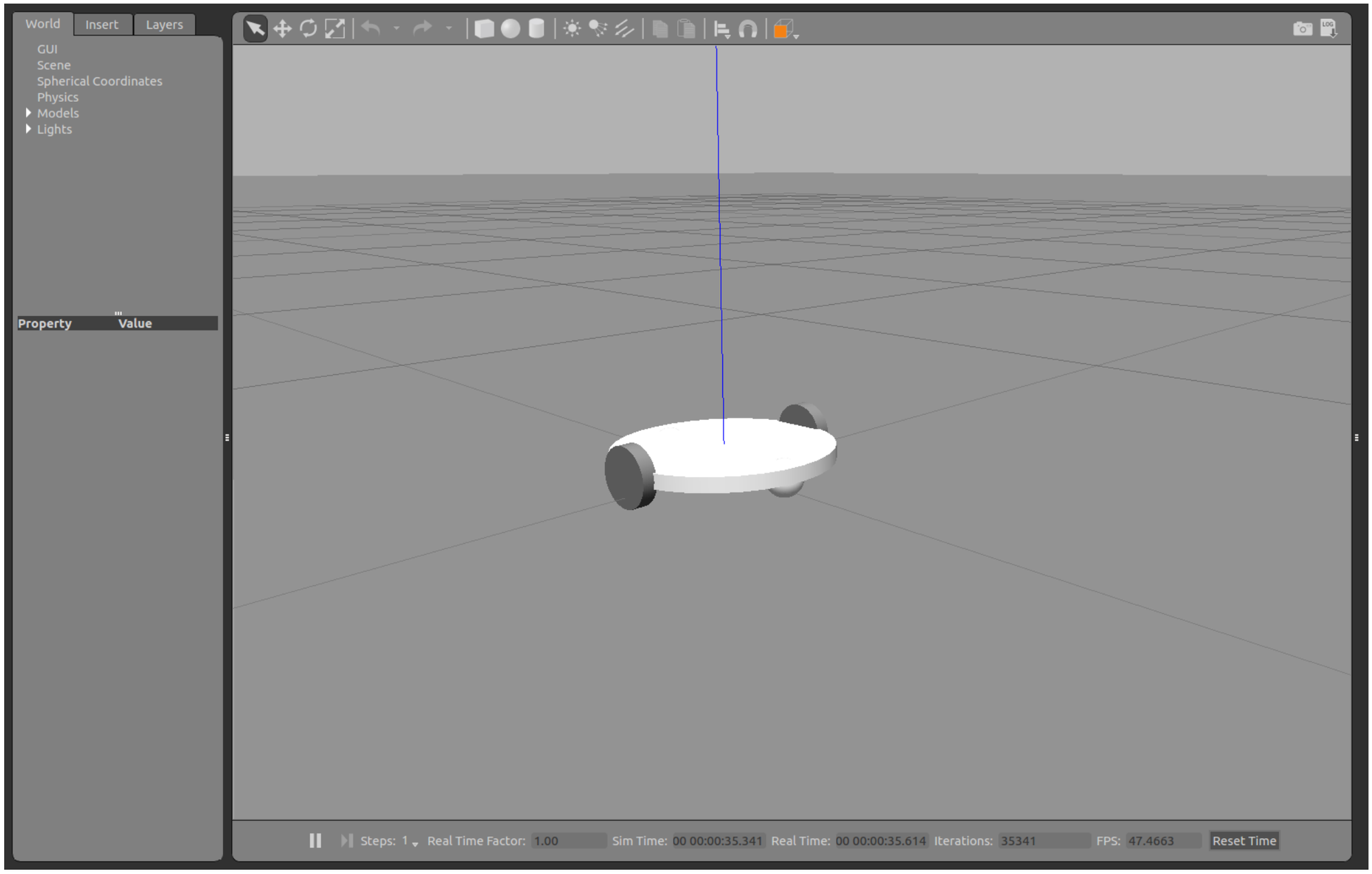
Load gazebo with an empty world

Spawn the robot inside the world

Simulating the Mobile Robot in Gazebo

Launch File

```
$ roslaunch model_tutorial mobile_base_gazebo.launch
```



Simulating the Mobile Robot in Gazebo

Adding the laser scanner to Gazebo

```
<!-- hokuyo -->
  …
<gazebo reference="hokuyo_link">
  <material>Gazebo/Blue</material>
  <turnGravityOff>false</turnGravityOff>
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>${hokuyo_size/2} 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>10.0</max>
        <resolution>0.001</resolution>
      </range>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Define the sensor characteristics

Load a plugin to simulate the laser scanner in ros

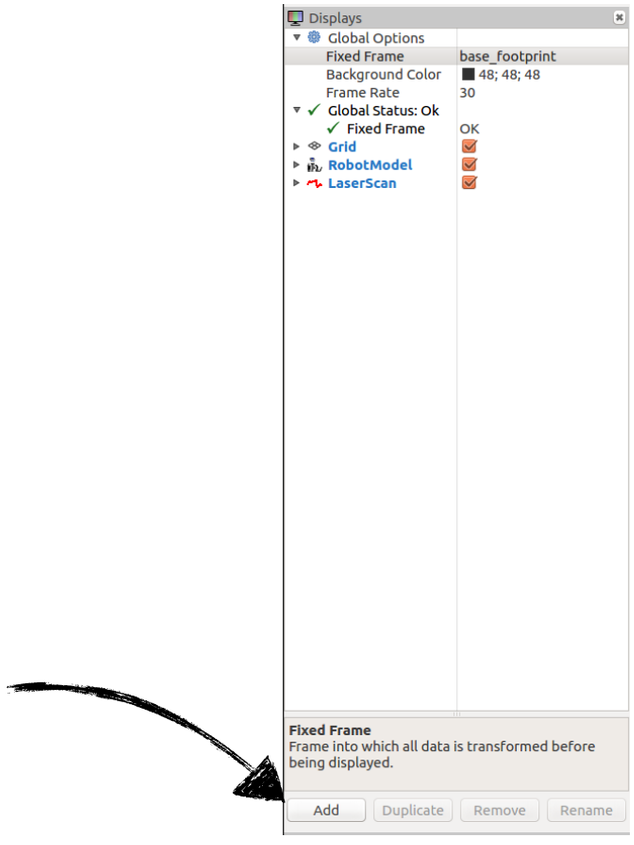
Simulating the Mobile Robot in Gazebo

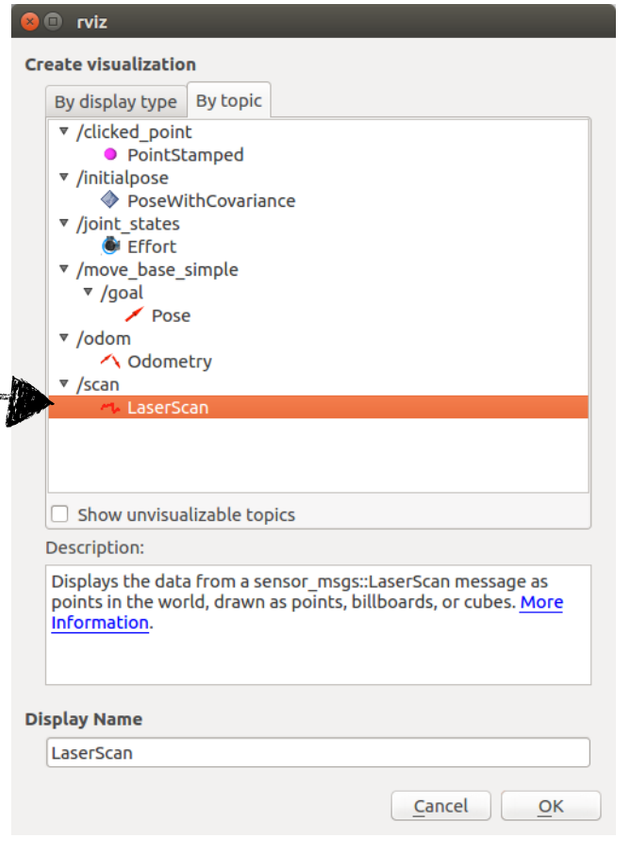
Visualize the simulated laser scanner in RViz

In two different consoles launch:

```
$ roslaunch model_tutorial mobile_base_gazebo.launch
$ roslaunch model_tutorial mobile_base.launch
```

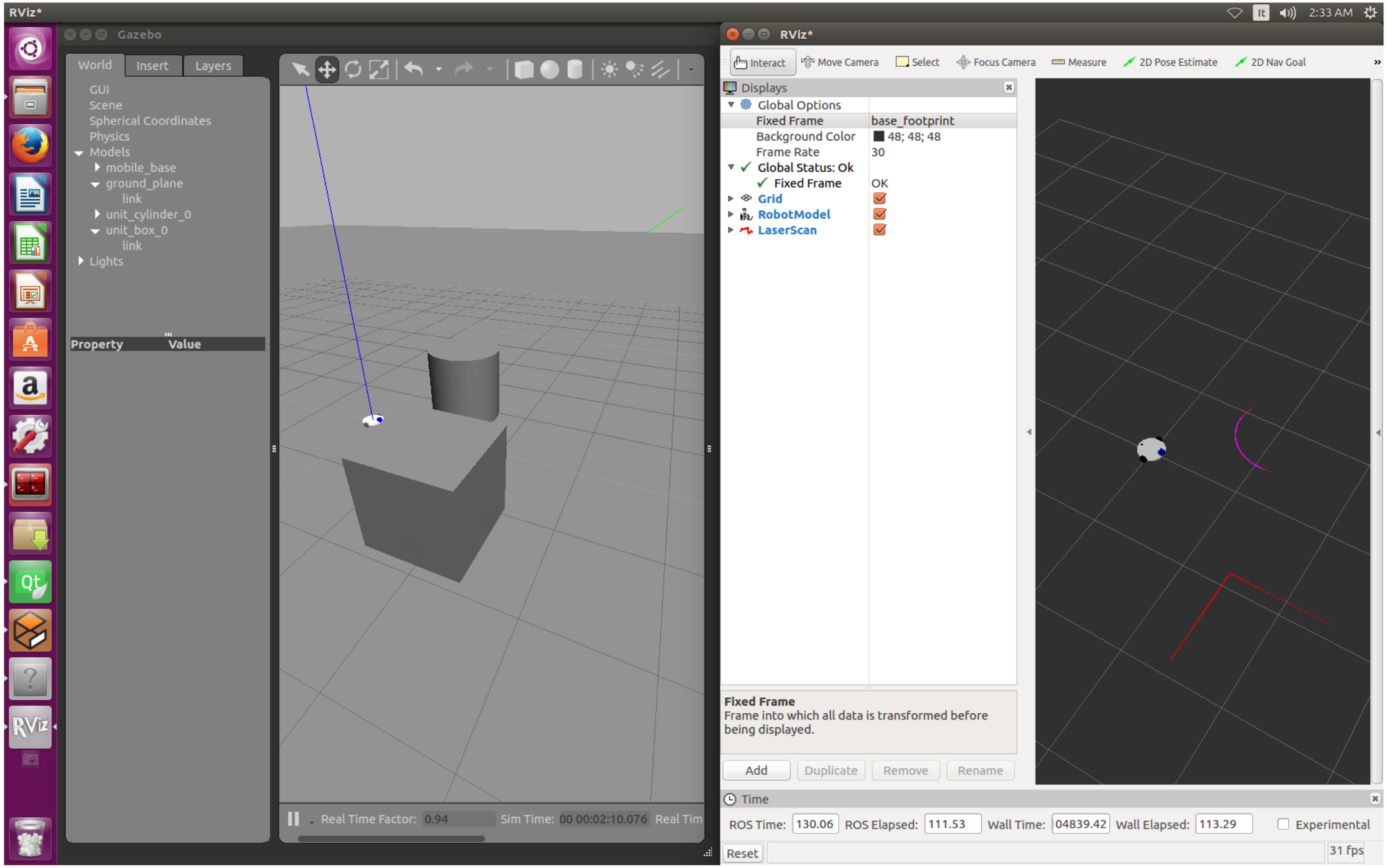
In RViz add the laser scan topic





Simulating the Mobile Robot in Gazebo

Visualize the simulated laser scanner in RViz



Simulating the Mobile Robot in Gazebo

Adding the differential drive controller to Gazebo

```
<!-- Differential drive controller  -->
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <rosDebugLevel>Debug</rosDebugLevel>
    <publishWheelTF>false</publishWheelTF>
    <robotNamespace>/</robotNamespace>
    <publishTf>1</publishTf>
    <publishWheelJointState>false</publishWheelJointState>
    <alwaysOn>true</alwaysOn>
    <updateRate>100.0</updateRate>
    <leftJoint>front_left_wheel_joint</leftJoint>
    <rightJoint>front_right_wheel_joint</rightJoint>
    <wheelSeparation>${2*base_radius}</wheelSeparation>
    <wheelDiameter>${2*wheel_radius}</wheelDiameter>
    <broadcastTF>1</broadcastTF>
    <wheelTorque>30</wheelTorque>
    <wheelAcceleration>1.8</wheelAcceleration>
    <commandTopic>cmd_vel</commandTopic>
    <odometryFrame>odom</odometryFrame>
    <odometryTopic>odom</odometryTopic>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <legacyMode>false</legacyMode>
  </plugin>
</gazebo>
```

Load a plugin to simulate a differential driver robot.

The plugin subscribe the cmd_vel command topic.