Chapter-3

Simulating Robot Using ROS and Gazebo

⇒ Robot simulation will give you an idea about the working of robots in a virtual environment.

Gazebo: Multirobot simulator for complex indoor and outdoor robotic simulation.

* **Some important packages**

- **gazebo_ros_pkgs**
  - → This contains wrappers and tools for interfacing ROS with Gazebo.
- **gazebo_msgs**
  - → This contains messages and Service data structures for interfacing with Gazebo from ROS.
- **gazebo_plugins**
  - → This contains Gazebo plugins for Sensors, actuators and so on.
- **gazebo_ros_control**
  - → This contains standard controllers to communicate between ROS and Gazebo.

★ Converting to Gazebo

Required

⇒ An <inertial> element within each <link> element must be properly specified and configured.

Optional

⊙ Add <gazebo> element for every <link>
- → Convert visual colors to Gazebo format.
- → add Sensor plugins

⊙ Add <gazebo> element for every <joint>
- → Set proper damping dynamics
- → Add actuator control plugins.

⊙ Add a <gazebo> element for the <robot> element

⊙ Add <link name="world"/> link if the robot should be rigidly attached to the world.

★ The <gazebo> Element

⇒ The <gazebo> element is an extension to the URDF used for specifying additional Properties needed for simulation purposes in Gazebo.

⇒ There are three different of <gazebo> elements:-
- → One for <robot> tag
- → One for <link> tag
- → One for <joint> tag

⇒ If a <gazebo> element is used without a reference Property, it is assumed the <gazebo> element is for the whole robot model.

★ Material in gazebo

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

★ <gazebo> Elements for link

- → Material
- → gravity (bool)
- → damping factor
- → maxVel
- → min Depth
- → mu1
- → mu2
- → fdir1
- → Kp
- → Kd
- → selfCollide
- → maxContacts
- → lesenRatio

★ <gazebo> Elements for Joints

- → StopCfm
- → StopErp
- → provideFeedback
- → implicitSpringDamper
- → CfmDamping
- → fudgeFactor

⇒ To Verify if your URDF can be properly converted into a SDF.

```
gz sdf -p MODEL.urdf
```

---

* Adding transmission tag to actuate the model

⇒ In order to actuate the robot using ROS controllers, we should define the <transmission> element to link actuators to joint.

```
<transmission name="tran1">
    <type> transmission_interface/SimpleTransmission
    </type>
    <joint name="joint_name">
        <hardwareInterface> PositionJointInterface
        </hardwareInterface>
    </joint>
    <actuator name="motor1">
        <mechanicalReduction> 1 </mechanicalReduction>
    </actuator>
</transmission>
```

⇒ Transmission <type>

↳ Currently, transmission_interface/SimpleTransmission is only supported.

⇒ The <hardwareInterface> element is the type of hardware interface to load (Position, Velocity, or effort interface)

↓

{ hardware interface is loaded by the gazebo_ros_control plugin }

* Adding the gazebo_ros_control plugin

↳ In order to parse the transmission tags and assign appropriate hardware interface and the control manager.

```
<gazebo>
  <Plugin name="gazebo_ros_control"
          filename="libgazebo_ros_control.so">
    <robotNamespace> / </robotNamespace>
  </Plugin>
</gazebo>
```

⇒ If we are not specifying the name, it will automatically load the name of the robot from the URDF.

⇒ We can also specify:

- Controller update rate — <controlPeriod>
- location of robot description on Parameter Server — <robotParam>
- Type of robot hardware interface — <robotSimType>
  - → JointStateInterface
  - → EffortJointInterface
  - → VelocityJointInterface

* Adding Sensor in Gazebo

⇒ To build a sensor in Gazebo, we have to model the behavior of that sensor in Gazebo.

↳ There are some prebuilt sensor models in Gazebo that can be used directly in our code without writing a new model.

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

  <gazebo reference="sensor_link">
    <sensor type="sensor type" name="name of sensor">
      ... Sensor Parameters ...
      <plugin name="name of plugin" filename="libgazebo_ros
                                              _name.so">
        ... Plugin parameters ...
      </plugin>
    </sensor>
  </gazebo>

</robot>
```

* <Sensor> tag for laser Scanner

```
<Sensor type="ray" name="name of sensor">
    <Pose> 0 0 0 0 0 0 </Pose>
    <Visualize> false </Visualize>
    <update_rate> 40 </update_rate>
```

⇒ frequency at which it gives new data

When true a semi-translucent laser ray is visualized within the scanning zone

Position and Orientation of Sensor

```
    <ray>
        <scan>
            <horizontal>
                <samples> 720 </samples>
                <min_angle> -1.57 </min_angle>
                <max_angle> 1.57 </max_angle>
            </horizontal>
        </scan>
        <range>
            <min> 0.10 </min>
            <max> 10.0 </max>
            <resolution> 0.001 </resolution>
        </range>
    </ray>
```

```
<Plugin name="Name of Controller"
        filename="libgazebo_ros_laser.so">
    <topicName> /scan </topicName>
    <frame.Name> Name of link </frame.Name>
</Plugin>

</sensor>
```

* Ros controllers

⇒ On each joint, we need to attach a Controller that is compatible with the hardware interface mentioned inside the transmission tag.

⇒ ROS Controller mainly consists of a feedback mechanism, most probably a PID loop, which can receive a set point, and control the output using the feedback from the actuator.

⇒ The ROS controllers are provided by a set of packages called ros_control.

⇒ The ros_control packages are composed of the following individual packages:

- control_toolbox
  - Contains common modules (PID & Sine) that can be used by all controllers.
- Controller_interface
  - Contains the interface base class for controllers.
- Controller_manager
  - Infrastructure to load, unload, start and stop controllers.
- Controller_manager_msgs
  - This provides the message and service definition for the controller manager.
- hardware_interface
  - Contains base class for the hardware interface.
- transmission_interface
  - Contains the interface classes for the transmission interface.
    (differential, four bar linkage, joint state, position and velocity)

* Different type of ROS Controllers and Hardware Interface

Joint_position_Controller
Joint_State_Controller
Joint_effort_Controller

Joint Command Interface
  # Effort Joint Interface
  # Velocity Joint Interface
  # Position Joint Interface
Joint State Interface