ECE 495/595 Lecture Slides

Winter 2017

Instructor: Micho Radovnikovich

Summary and Quick Links

These slides contain the following concepts:

- Augmenting a URDF model for Gazebo (Slide 3)
- Collision elements (Slide 4)
- Inertial elements (Slide 6)
- Gazebo specific properties (Slide 9)
- Setting up ros_control (Slide 12)
- Spawning the model and controllers in Gazebo (Slide 16)

Augmenting a URDF model for Gazebo

- To extend a visual robot model into a Gazebo simulation model, more properties must be set.
- These include collision geometry, inertial properties, and Gazebo specific parameters such as friction coefficients.
- To simulate joint actuators, **ros_control** is used, which requires more modifications to the URDF model.
- Special nodes need to be run to spawn the model and the joint controllers in Gazebo

Collision Elements

- Each link in a URDF model can have any number of collision elements.
- A link's collision elements define geometric shapes that will be used by Gazebo to detect collisions with other objects in the simulated world.
- Collision geometries do not have to be the same shape as the corresponding visual elements of the link.
- Typically, simple shapes are used for collision to minimize the computational complexity of the collision detection process.

Collision Elements

- To specify collision geometry in the URDF model, the **<collision>** element is used.
- A **<collision>** element is identical to the **<visual>** element, except that material properties can't be set.

```
<collision>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <geometry>
    <box size="1.0 0.5 1.0" />
  </geometry>
</collision>
```

Inertial Elements

- Each link can also have any number of inertial elements, which are used to specify mass and moment of inertia tensors.
- When loading a URDF model in Gazebo, every link must have at least one inertial element, even if there are no visual or collision geometries specified for that link.
- The mass of a link is specified in a <mass> element.

Inertial Elements

- The rotational mass of a robot link is specified in an **inertia tensor**.

$$I = \begin{bmatrix} i_{xx} & i_{xy} & i_{xz} \\ i_{yx} & i_{yy} & i_{yz} \\ i_{zx} & i_{zy} & i_{zz} \end{bmatrix}$$

- The inertia tensor is a matrix that relates angular momentum to the angular velocity of a rigid body.

$$H = \begin{bmatrix} i_{xx} & i_{xy} & i_{xz} \\ i_{yx} & i_{yy} & i_{yz} \\ i_{zx} & i_{zy} & i_{zz} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

Inertial Elements

- The inertia tensor is a symmetric matrix

$$i_{xy} = i_{yx} \qquad i_{xz} = i_{zx} \qquad i_{yz} = i_{zy}$$

- Therefore, only 6 parameters are specified in the URDF model to describe the inertia tensor.

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

Gazebo Specific Properties

- For a robot model to look and operate properly in Gazebo, some special properties must be set that are not required for a visual model.
- There are several properties defined in Gazebo, but the most important ones for making a functional vehicle are the friction properties.
- To specify a Gazebo property instead of a regular URDF property, they must all be wrapped in a **<gazebo>** element.
- If the properties are applied to a particular link, the link name is specified as a **reference** in the **<gazebo>** element.

Gazebo Specific Properties

- Friction coefficients are specified using the **<mu1>** and **<mu2>** tags.
- **mu1** is the coefficient of friction along the primary axis of whichever collision geometry is used for the particular link.
- **mu2** is the coefficient of friction along the secondary axis.

```
<gazebo reference="link_name" >
  <mu1>1.0</mu1>
  <mu2>1.0</mu2>
</gazebo>
```

Gazebo Specific Properties

- Gazebo cannot use the material colors specified in a **<material>** element, as in a visual model.
- Instead, Gazebo colors are defined using OGRE material scripts.
- Pre-defined colors are available, or custom OGRE materials can be created.

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

Setting Up ros_control

- Actuation of the joints of a URDF robot model can be simulated using **ros_control**.
- The control interface for a joint can be position, velocity, or effort based. For revolute joints, effort is the torque, and for prismatic joints, effort is the linear force.
- The mechanical interface between actuators and joints can be described using **transmissions**.
- **ros_control** properties are specified in the URDF model.

Setting Up ros_control

- Load the **gazebo_ros_control** plugin.

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

- Specify a **transmission** for each joint that actuates.

```
<transmission name="left_control">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="left_wheel_joint" >
    <hardwareInterface>VelocityJointInterface </hardwareInterface>
  </joint>
  <actuator name="left_wheel_actuator">
    <hardwareInterface>EffortJointInterface </hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Setting Up ros_control

▷ Parameters for roscontrol are set in a YAML file.

```yaml
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

left_wheel_controller:
  type: velocity_controllers/JointVelocityController
  joint: left_wheel_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}

right_wheel_controller:
  type: velocity_controllers/JointVelocityController
  joint: right_wheel_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}
```

Setting Up roscontrol

- The namespaces specified in the YAML file are the names of the controllers, and are arbitrary.
- The **joint** parameter inside of a controller namespace refers to the joint specified in a **transmission** tag.
- The PID gain parameters are what the **ros_control** plugin will use to perform closed loop control of the simulated actuators.

Spawning Robot Model and Controllers

- The robot model is spawned by running the **spawn_model** node in the **gazebo_ros** package.

```
<node pkg="gazebo_ros" type="spawn_model"  name="spawn_mantis"
     args="-urdf -param robot_description -model mantis"/>
```

- The controllers are spawned by running the **spawner** node in the **controller_manager** package.

```
<node pkg="controller_manager" type="spawner" name="controller_spawner"
  args="left_wheel_controller right_wheel_controller
            joint_state_controller --shutdown-timeout 0.5"/>
```

Spawning Robot Model and Controllers

- The **args** for the model spawner are:
  - **-urdf**: Indicates that it is a URDF model being loaded.
  - **-param**: This is the parameter that contains the description of the robot.
  - **-model**: This is the name of the model that will be spawned in Gazebo.

Spawning Robot Model and Controllers

- The **args** for the controller spawner are:
  - A space-separated list of the different **ros_control** modules to load. These should match the namespace names in the YAML file.
  - **shutdown-timeout**: This is how long the controllers wait to shutdown when a connection to the simulated actuators is lost. It defaults to 30 seconds.