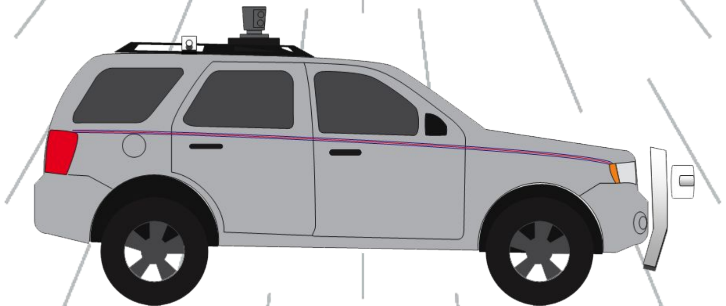
THE UNIVERSITY OF ARIZONA

CAT VEHICLE REU 2019

Robot Design using ROS and Gazebo

Rahul Bhadani <rahulbhadani@email.arizona.edu>



Download the slides from
https://rahulbhadani.github.io/reu.html

Agenda

- URDF - The **U**niversal **R**obot **D**escription **F**ormat
- Joints and Links
- Writing a XACRO file
- Model in Gazebo
- World in Gazebo
- Understanding physics in Gazebo
- Plugin to manipulate Robot state in Gazebo
- Sensor simulation
- Integrating ROS with Gazebo

URDF-Universal Robot Description Format

- Although not an universal format, it is adopted by ROS to specify robot structures.
- An old format that doesn't address very well evolving needs of Robots
- Specifies only Kinematic and Dynamic Properties
- Cannot specify pose of the model related to the world.
- **But this is what we have got for now.**
- **Later, we will see that sdf file format tries to fill some gaps.**

URDF: First Example

- Create a file and enter the following content:

```
<?xml version="1.0"?>
<robot name="myfirst">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>
</robot>
```

- Save it as **first.urdf** and close it.

- Run the following command in the terminal

```
$ roslaunch urdf_tutorial display.launch model:=first.urdf
```

URDF: First Example

- Adding more stuff:
  - So far we had a primitive shape (called as **link**): a cylinder. How do we we create a complicated body?
  - Consider a complicated body as a combination of many primitive shapes.
  - How are they connected?
  -

URDF: First Example

- Add another primitive shape (i.e. link).
- Two links are connected by **joints.**
- Added following content in a new file: **second.urdf**

```
<?xml version="1.0"?>
<robot name="multipleshapes">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>
```

```
<link name="right_leg">
  <visual>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
  </visual>
</link>
<joint name="base_to_right_leg" type="fixed">
  <parent link="base_link"/>
  <child link="right_leg"/>
</joint>
</robot>
```

Run as `roslaunch urdf_tutorial display.launch model:=second.urdf`

URDF: First Example

- Notice that in the last example, we didn't specify, how will two links be connected: from center to center, from edge to center of from edge to edge?
- By default if we do not specify that, then joints connect two links by their centers (of mass).
- In order to connect two links differently, we can define origins explicitly.

URDF: First Example

- Create a new file: third.urdf

```
<?xml version="1.0"?>
<robot name="origins">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6"
radius="0.2"/>
      </geometry>
    </visual>
  </link>
```

```
 <link name="right_leg">
    <visual>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0
-0.3"/>
    </visual>
  </link>
  <joint name="base_to_right_leg"
type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0 -0.22 0.25"/>
  </joint>
</robot>
```

URDF: First Example

- Examine the joint's origin `<origin xyz="0 -0.22 0.25"/>`.
- It is defined in terms of the parent's reference frame.
- We are -0.22 meters in the y direction (to our left, but to the right relative to the axes) and 0.25 meters in the z direction (up).
- This means that the origin for the child link will be up and to the right, regardless of the child link's visual origin tag.
- Since we didn't specify a rpy (roll pitch yaw) attribute, the child frame will be default, i.e., have the same orientation as the parent frame.

URDF: First Example

- Now, looking at the leg's visual origin, <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
- It has both a xyz and rpy offset. This defines where the center of the visual element should be, relative to its origin.
- We want the leg to attach at the top, we offset the origin down by setting the z offset to be -0.3 meters.
- Since we want the long part of the leg to be parallel to the z axis, we rotate the visual part PI/2 around the Y axis.

URDF: First Example

- We can add few other attributes like colors, material type, texture etc.

URDF: First Example

```
<?xml version="1.0"?>
<robot name="materials">
  <material name="blue">
    <color rgba="0 0 0.8 1"/>
  </material>
  <material name="white">
    <color rgba="1 1 1 1"/>
  </material>
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue"/>
    </visual>
  </link>
```

```
<link name="right_leg">
  <visual>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
    <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
    <material name="white"/>
  </visual>
 </link>
<joint name="base_to_right_leg" type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0 -0.22 0.25"/>
  </joint>
```

```
  <link name="left_leg">
    <visual>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0
-0.3"/>
      <material name="white"/>
    </visual>
   </link>

   <joint name="base_to_left_leg" type="fixed">
     <parent link="base_link"/>
     <child link="left_leg"/>
     <origin xyz="0 0.22 0.25"/>
   </joint>

</robot>
```

URDF: First Example

- A Full Example can be obtained from: https://github.com/ros/urdf_tutorial/blob/master/urdf/05-visual.urdf
-

URDF: Specifying joint types

- Use the example from https://github.com/ros/urdf_tutorial/blob/master/urdf/06-flexible.urdf
- Joints are used to move different links
- Types of joints:
  - revolute - a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
  - continuous - a continuous hinge joint that rotates around the axis and has no upper and lower limits.
  - prismatic - a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
  - fixed - This is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the axis, calibration, dynamics, limits or safety_controller.
  - floating - This joint allows motion for all 6 degrees of freedom.
  - planar - This joint allows motion in a plane perpendicular to the axis.

URDF: Specifying joint types

- We will examine few components of the code.

```
<joint name="head_swivel" type="continuous">
   <parent link="base_link"/>
   <child link="head"/>
   <axis xyz="0 0 1"/>
   <origin xyz="0 0 0.3"/>
 </joint>
```

- We see that connection between body and head is continuous, so head can freely rotate.

THE UNIVERSITY OF ARIZONA

URDF: Specifying joint types

```
<joint name="left_gripper_joint" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="1000.0" lower="0.0" upper="0.548" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
    <parent link="gripper_pole"/>
    <child link="left_gripper"/>
  </joint>
```

- Arms rotate using revolute. Revolute joints are same as continuous but with some restriction on degree of movement.
-

URDF: Specifying joint types

```
<joint name="gripper_extension" type="prismatic">
    <parent link="base_link"/>
    <child link="gripper_pole"/>
    <limit effort="1000.0" lower="-0.38" upper="0" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0.19 0 0.2"/>
  </joint>
```

- Gripper Extension uses prismatic joints. It moves along an axis, not around it. This translational movement is what allows our robot model to extend and retract its gripper arm.

URDF: Adding physical properties

- Until now, we specified structure of the robot. We will not add some physical attributes
- Use the file from https://raw.githubusercontent.com/ros/urdf_tutorial/master/urdf/07-physics.urdf
-

URDF: Adding physical properties

```
<link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 .8 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </collision>
  </link>
```

- The collision element defines its shape the same way the visual element does, with a geometry tag. The format for the geometry tag is exactly the same here as with the visual.
- Collision defines what shape will be used for physics calculation as in force computations when two bodies (or links) touch or collide with each other.
- Usually, visual geometry can be a complicated mesh developed in solidworks or similar software, but collision is kept simple as a mesh collision makes computation slow and time-expensive.

URDF: Adding physical properties

- We also need to specify physical properties such inertia, mass, friction, etc that physics engines (e.g. Gazebo) would need.
- Meshlab software can be used to calculate inertia of a complicated geometry.
- Wrong inertia will make everything going haywire in Gazebo.

```
 <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 .8 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="10"/>
      <inertia ixx="0.4" ixy="0.0" ixz="0.0"
iyy="0.4" iyz="0.0" izz="0.2"/>
    </inertial>
  </link>
```

URDF: Using XACRO in URDF

- XACRO is a macro language, used with URDF to define variables and do computations within URDF to avoid doing math by hand.
- Open your **catvehicle.xacro** file from catvehicle package and examine the content of the file
-

Using URDF in Gazebo

- As we discussed earlier, urdf comes with its own shortcomings.
- To fill the gaps for evolving needs of robots, a new format called the Simulation Description Format (SDF) was created for use in Gazebo
- Run the command: `roslaunch urdf_sim_tutorial gazebo.launch`
- You will see a robot in the Gazebo world

Using URDF in Gazebo

- Now we examine what is it in the `gazebo.launch`
- To find the `gazebo.launch` file, type `roscd urdf_sim_tutorial` in your terminal.
- roscd command changes directory that has package `urdf_sim_tutorial`
- `cd launch`
- `gedit gazebo.launch`

Using URDF in Gazebo

- Now, run `roslaunch urdf_sim_tutorial 13-diffdrive.launch`
- Play around. See robot moving.
- Examine some files:
    - `roscd urdf_sim_tutorial`
    - `cd launch`
    - `gedit 13-diffdrive.launch`

USING MATLAB to examine urdf

- In MATLAB type:
- `robot = importrobot('sixth.urdf');`
- `show(robot);`

- You can also use Simscape in MATLAB to analyze and perform simulations on urdf file.
- `smimport('sixth.urdf');`

Exercise

- Follow the tutorial on
- http://gazebosim.org/tutorials/?tut=ros_urdf
- http://gazebosim.org/tutorials?tut=ros_gzplugins
- http://gazebosim.org/tutorials/?tut=ros_control