ROS Made Easy
5: Gazebo

Aniruddh K Budhgavi
Enigma, IIIT-B

June 26, 2020

1 Notes

1. This tutorial was created for **ROS1 Melodic Morenia** on **Ubuntu 18.04 Bionic Beaver**, in **June 2020**. I expect them to become rapidly out of date. It is my hope that Team Enigma will continually maintain and update these tutorials.
2. This tutorial assumes that you are running Ubuntu, and have at least an elementary grasp of Python 2.7 and C/C++ .
3. All the code and the models for this tutorial are available at `https://github.com/aniruddhkb/enigmatutorials`.
4. The aim of this tutorial is to make you *functional* in ROS, not to make you a master. For that, look elsewhere.
5. Ensure that you have RViz, TF2, the robot_state_publisher package and the joint_state_publisher and joint_state_publisher_gui packages installed before proceeding further. You can install them using the relevant `apt-get` commands. See `http://wiki.ros.org/rviz/UserGuide`, this and this.
6. Ensure that you have **Meshlab** installed. Get it here.

2 Introduction

1. Gazebo is an industry-standard robotics simulation tool. It integrates well with ROS, however, its compatibility with the URDF file format is a little fuzzy – it uses the SDF file format by default instead. To make our URDF file Gazebo-compliant, we must add additional information to our URDF. The most critical information we must provide are the *inertial* and *collision* properties of our links.
2. To add the collision properties, add something like the below code between your `link` tags:

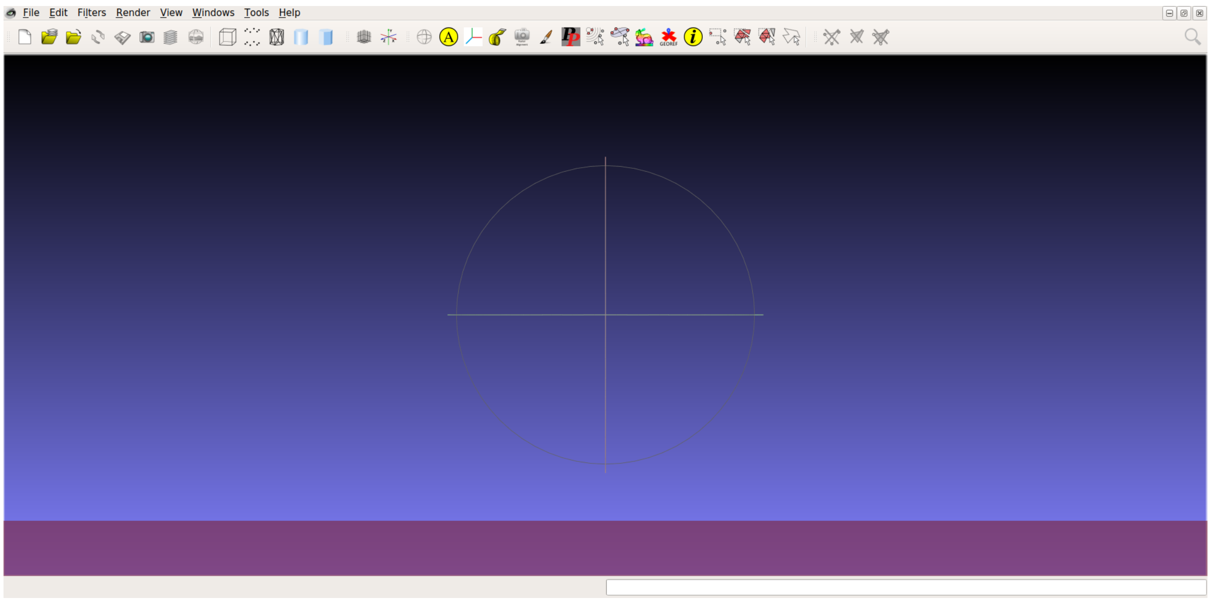
```
<collision>
  <geometry>
    <mesh filename="package://turret_bot/meshes/base.stl" scale="1 1 1"/>
  </geometry>
</collision>
```

   Add the corresponding meshes to the links. Here, we are using the same mesh files as our `visual` meshes. This is fine for this example, however, typically, one uses complex and visually appealing meshes for `visual` and greatly simplified meshes (or even basic shapes) for the `collision` meshes.
3. Next, we must add the *inertial* properties to our model. To compute these inertial properties, we use a software called **Meshlab**. You can get Meshlab from `https://snapcraft.io/install/meshlab/ubuntu`.

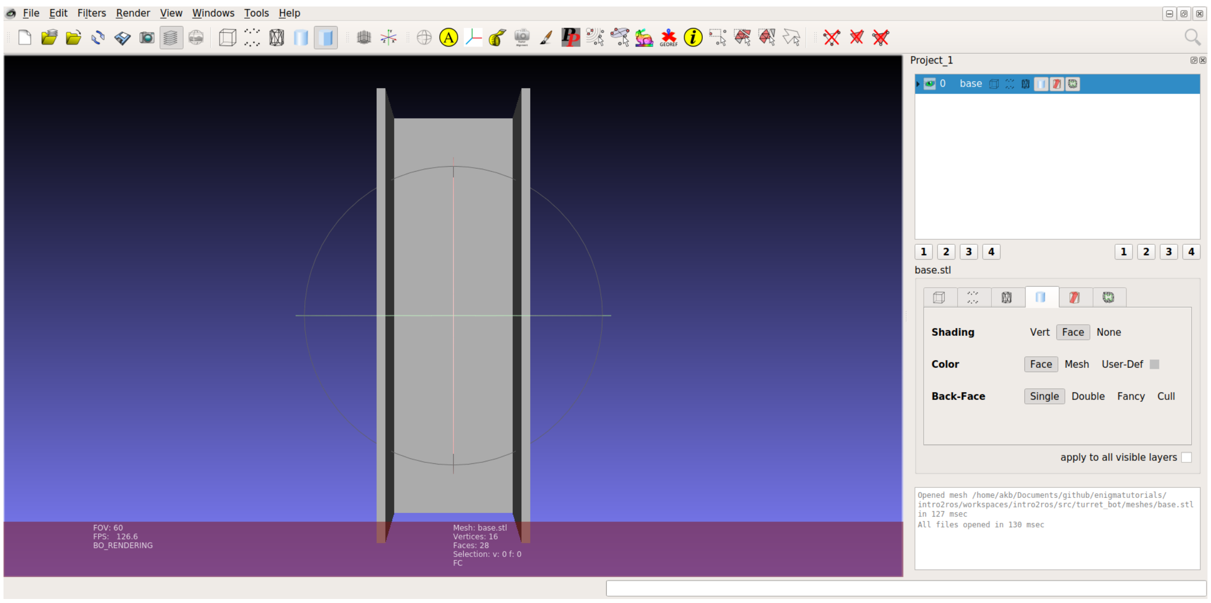
3 Inertial properties

3.1 Meshlab

1. Launch Meshlab. You should see the following window:



2. File – Import Mesh – base.stl . You should see:



```
FOV: 60
FPS:   126.6
BO_RENDERING
```

```
Mesh: base.stl
Vertices: 16
Faces: 28
Selection: v: 0 f: 0
FC
```

Shading: Vert [✓ Face] None
Color: [✓ Face] Mesh User-Def
Back-Face: [✓ Single] Double Fancy Cull

```
Opened mesh /home/akb/Documents/github/enigmatutorials/
intro2ros/workspaces/intro2ros/src/turret_bot/meshes/base.stl
in 127 msec
All files opened in 130 msec
```

3. Filters – Quality Measure and Computations – Compute Geometric Measures. The output (bottom right corner box):

```
Opened mesh base.stl in 127 msec
All files opened in 130 msec
Mesh Bounding Box Size 0.340000 1.000000 0.170000
Mesh Bounding Box Diag 1.069813
Mesh Bounding Box min -0.170000 0.000000 -0.020000
Mesh Bounding Box max 0.170000 1.000000 0.150000
Mesh Surface Area is 1.345600
Mesh Total Len of 42 Edges is 20.151415 Avg Len 0.479796
```

```
    Thin shell (faces) barycenter: 0.000000 0.500000 0.033118
    Vertices barycenter 0.000000 0.500000 0.070000
    Mesh Volume is 0.012800
    Center of Mass is 0.000000 0.500000 0.029844
    Inertia Tensor is :
    | 0.001101 0.000000 0.000000 |
    | 0.000000 0.000254 -0.000000 |
    | 0.000000 -0.000000 0.001286 |
    Principal axes are :
    | 0.000000 1.000000 0.000000 |
    | 1.000000 0.000000 0.000000 |
    | 0.000000 0.000000 1.000000 |
    axis momenta are :
    | 0.000254 0.001101 0.001286 |
    Applied filter Compute Geometric Measures in 42 msec
```

Note down the *mesh volume* and the *center of mass*. Have a look at the CoM value and ensure that the dimensions make sense. If they don't, you probably exported the mesh in millimeters instead of meters.

4. Filters – Normals, Curvatures and Orientation – Transform: Scale, Normalize – X Axis: 10, Y Axis: 10, Z Axis: 10

5. Run Compute Geometric Measures again. This time, note down the Inertia Tensor. **This is not the final inertia tensor**(yet).

```
    Inertia Tensor is :
    | 110.117317 0.000000 0.000000 |
    | 0.000000 25.381306 -0.000000 |
    | 0.000000 -0.000000 128.597351 |
```

6. Now, to get the real inertia tensor, multiply this matrix by the density of the material in **SI units** and 10^{-3}. Further, to get the mass, multiply the volume you have above by the density in SI units. I have written a short Jupyter notebook to assist me in these computations. It is available at the Github repo.

7. Do this for all the links in your model. The reason we went through that step of rescaling and rerunning the geometric measures computation is so that we could get more decimal places in our calculation of the inertia tensor. For more information, see `http://gazebosim.org/tutorials?tut=inertia&cat=build_robot`.

3.2 What is the inertia tensor?

1. At this stage, you must be having some questions about Meshlab and the inertia tensor.

2. Recall that the rotational equivalent for mass is the moment of inertia about the axis of rotation.

3. This definition of rotational inertia is useful, but is limited because it is only applicable for a particular axis of rotation (ie a particular joint). In a robot, a body may be simultaneously rotating about multiple axes in different directions.

4. The inertia tensor is the answer to this dilemma. The inertia tensor is a generalization of the concept of moment of inertia to three dimensions.

5. More formally, the inertia tensor about a point about a set of axes is defined as:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} \sum m_i(y_i^2+z_i^2) & -\sum m_i x_i y_i & -\sum m_i x_i z_i \\ -\sum m_i x_i y_i & \sum m_i(x_i^2+y_i^2) & -\sum m_i y_i z_i \\ -\sum m_i x_i z_i & -\sum m_i y_i z_i & \sum m_i(x_i^2+y_i^2) \end{bmatrix}$$

6. The inertia tensor provided by Meshlab is about the specified axes about the center of mass. The assumption is that the body has a homogenous mass distribution of unit density, which is why we need to multiply the density in SI units to get the correct answer.

7. For more information, see `https://en.wikipedia.org/wiki/Moment_of_inertia`

3.3 The inertia tag

1. Add something like the below code to each of your links:

```
    <inertial>
        <origin xyz="0 0.5 0" rpy="0 0 0"/>
        <mass value="16"/>
        <inertia ixx="1.376" ixy="0" ixz="0"
                 iyy="0.317" iyz="0"
                 izz="1.607"/>
    </inertial>
```

2. `origin` gives the center of mass.

3. `mass` is in kilograms (yes, that's a high mass. We usually use hollow bodies for this reason).

4. `inertia` is the inertia tensor, expressed using the six numbers needed to specify it as given above.

4 Further steps

4.1 Materials

Gazebo cannot use your `visual/material` tags. In order to specify colors for the model, add the following between the `robot` tags of the URDF.

```
<gazebo reference="base">
<material>Gazebo/Green</material>
</gazebo>
```

Add this for each of your links. For a list of Gazebo materials, see this page.

4.2 Joints

1. Ensure that your joint limits are correct. You'll get incorrect or unexpected behaviour otherwise. If you wish to use position control later on, be sure that the joint limits are non-overlapping – otherwise there will be trouble.

2. Add the following to each joint:

```
    <dynamics damping="0.001" friction="0.001"/>
```

   (a) `damping` is the viscous damping in Nm/(rad/s) or N/(m/s).
   (b) `friction` is the static frictional force in Nm or N.

4.3 The config file

In the package directory (very important – not in a subfolder), create `model.config`:

```
<?xml version="1.0"?>
<model>
    <name>turret_bot</name>
    <version>1.0</version>
    <sdf>urdf/turret_bot.urdf</sdf>
    <author>
        <name>Aniruddh K B </name>
        <email>aniruddhkb@gmail.com</email>
    </author>
    <description>
    A simple 2-dof robot for demonstration purposes.
    </description>
</model>
```

This is just a requirement for Gazebo.

4.4 The launch file

Create `launch/gazebo_spawn.launch`:

```
<?xml version="1.0"?>
<launch>
    <param name = "/robot_description"
     textfile="$(find turret_bot)/urdf/turret_bot.urdf"/>
    <include file = "$(find gazebo_ros)/launch/empty_world.launch"/>
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
        args=" -unpause -urdf -model turret_bot
               -param robot_description " respawn="false"/>
</launch>
```

4.5 Modify the environment variables

Add something like the following to your `.bashrc` :

```
export GAZEBO_MODEL_PATH=/usr/share/gazebo-9/models
                         :<workspace-path>/src
```

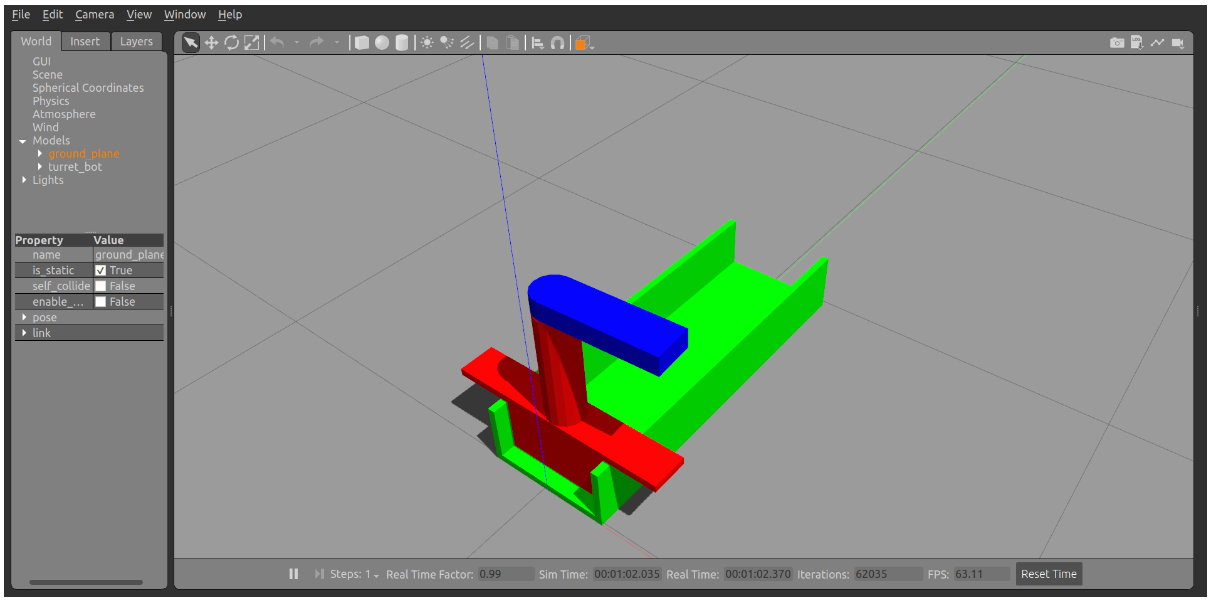
This is so that Gazebo can find your models.

4.6 Fixing a bug

There is a bug in Gazebo which must be fixed before the first time you launch Gazebo. Open `/.ignition/fuel/config.yaml` and modify `https://api.ignitionfuel.org` to `https://api.ignitionrobotics.org`.

5 Moment of truth

1. Launch the launch file you created just now. If all goes well, you should see something like:

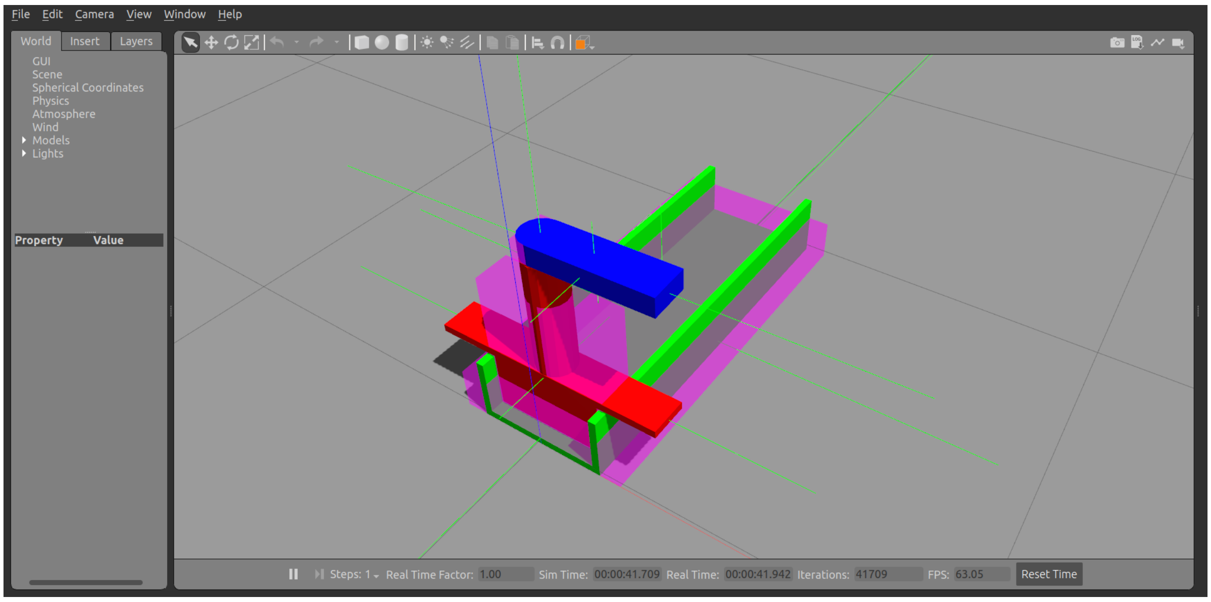


Property	Value
name	ground_plane
is_static	True
self_collide	False
enable_...	False
pose	
link	

Steps: 1 Real Time Factor: 0.99 Sim Time: 00:01:02.035 Real Time: 00:01:02.370 Iterations: 62035 FPS: 63.11 Reset Time

   Use the three mouse buttons to navigate the world.

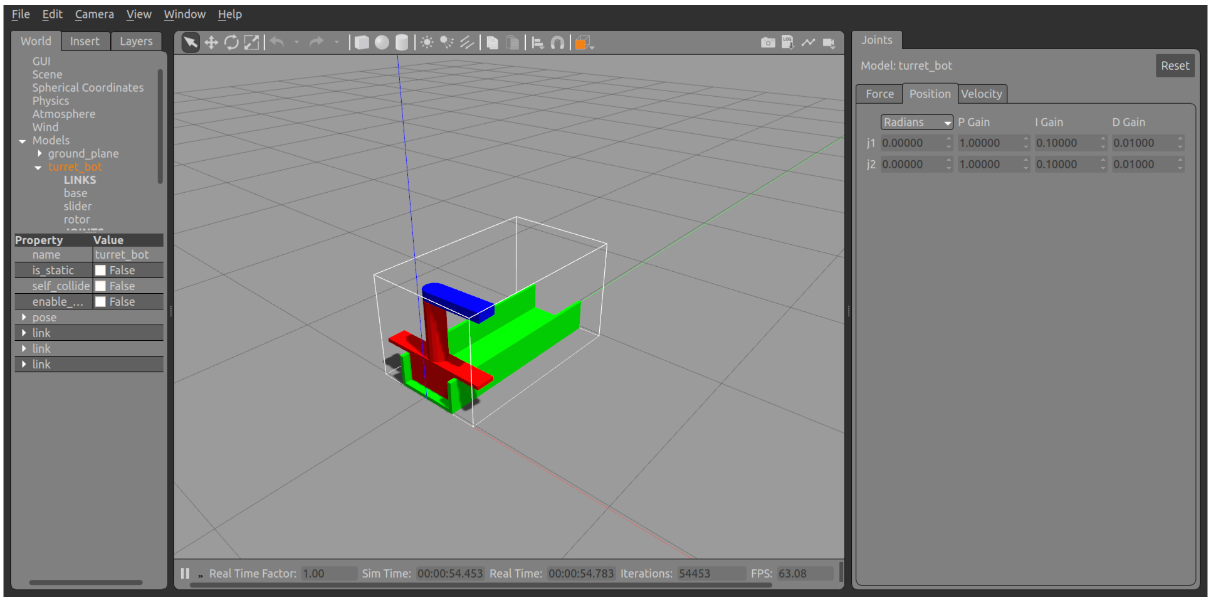
2. View – Moment of Inertia should show something like:



Steps: 1 Real Time Factor: 1.00 Sim Time: 00:00:41.709 Real Time: 00:00:41.942 Iterations: 41709 FPS: 63.05 Reset Time

   Those purple boxes are a representation of the inertia of those bodies. Ideally, they should be pretty close (in terms of size) to the actual link size.

3. Deactivate the inertia view, and open the right side pane by pulling from the right of the window. Click on the model and choose ”position”. You should see:



Property	Value
name	turret_bot
is_static	☐ False
self_collide	☐ False
enable_...	☐ False
pose	
link	
link	
link	

Model: turret_bot

	Radians	P Gain	I Gain	D Gain
j1	0.00000	1.00000	0.10000	0.01000
j2	0.00000	1.00000	0.10000	0.01000

4. You can actuate the robot by giving position values to the joints. Be warned, however. The position control is done through a PID feedback loop – you'll have to tune the gains for P, I and D properly in order to get good results. See this video for the process I followed (it's fourteen minutes long, so you can skip along once you get a hang of it). This is a good intro to PID tuning.

6 Looking ahead

You can now simulate a simple robot using Gazebo. Next time, we will use the `gazebo_ros_control` plugin to control our simulated robot using ROS.