ROS Made Easy
4: URDF and RViz

Aniruddh K Budhgavi
Enigma, IIIT-B

June 25, 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 I expect them to become rapidly out of date.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.

2 Designing a robot in Fusion 360

2.1 Getting Fusion 360

1. Fusion 360 is a 3D design and modelling package provided by Autodesk. Unfortunately, ROS/Gazebo work best in Linux while Fusion 360 is only available for Windows/Mac, so you'll have to design in Windows/Mac and simulate in Linux.

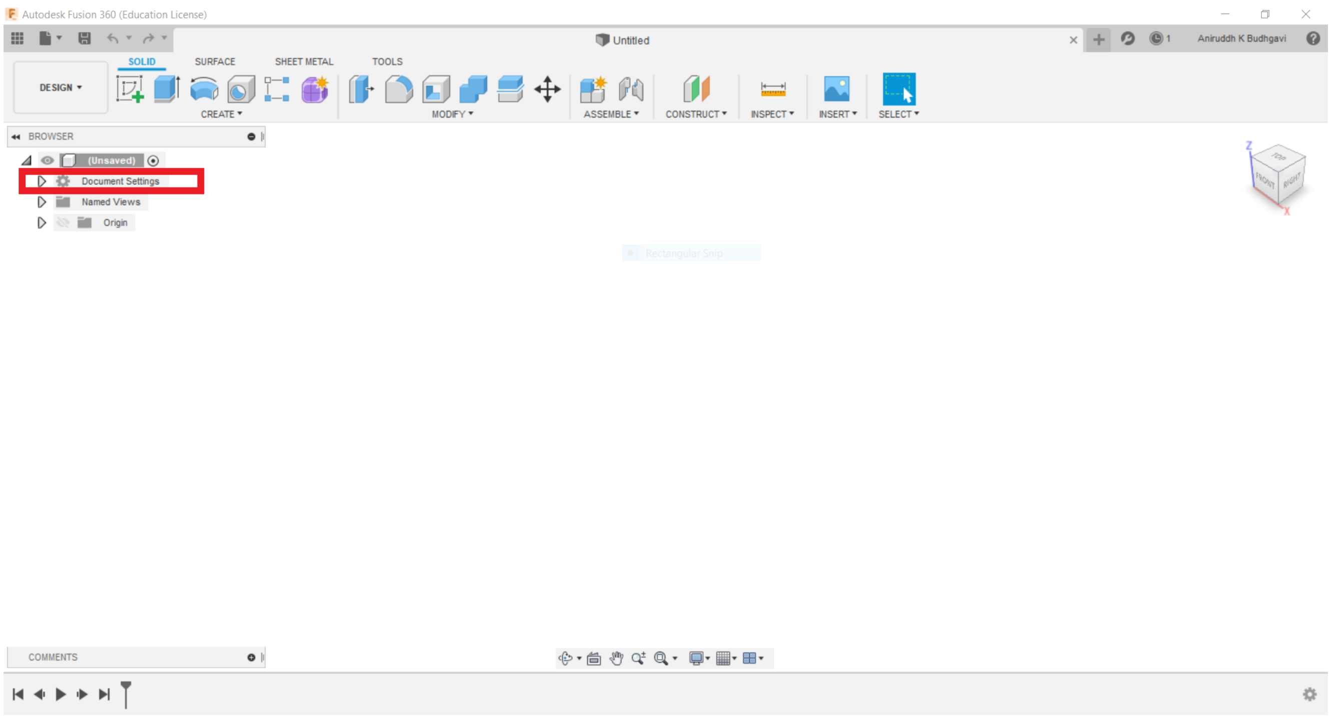
2. If you are a student or educator, you can get Fusion 360 for free at this link.

3. You can learn how to use Fusion 360 through the many Youtube tutorials on the topic. Mine is here. This is just one hour long, but covers many of the tools you will need to design models effectively in Fusion.

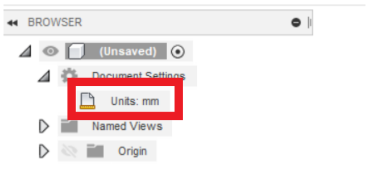
2.2 Modelling and exporting

1. First, change the model units from millimeters to meters. This is important because we will be exporting the model files as .STL files. The thing about STL files is that they are completely unitless in nature. It is up to the software to interpret the STL appropriately. Therefore, if the units of the source and destination software don't match, we will have an anomaly in the size of the model in the destination software. To do this:

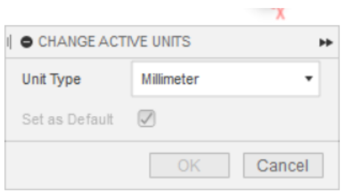
   (a) See the document settings option.



   (b) See the units option.

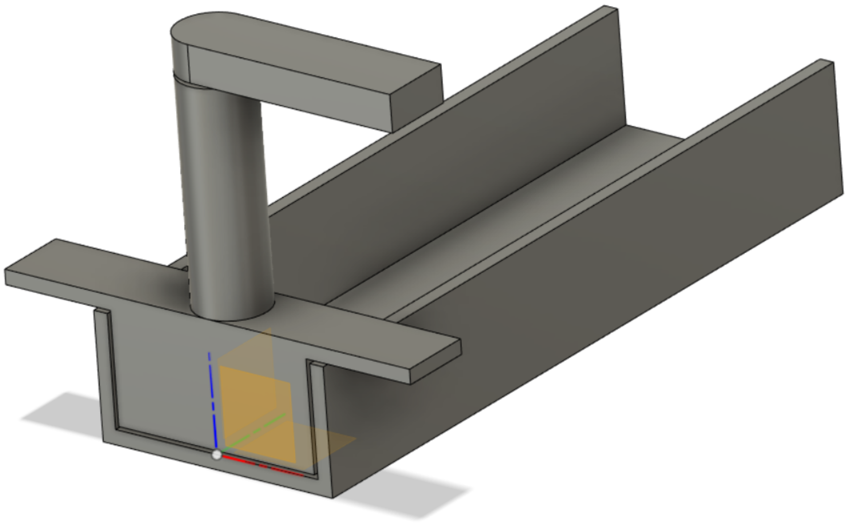


(c) Change the active units from mm to m.



Unit Type: Millimeter

2. Now, you can freely design the model. Segment the model into multiple independent bodies (the links of the robot) and leave small gaps between the bodies. Your model could look something like this:



3. Convert each body into a component and save it in its own file (The right-click context menu on an entry in the left-side browser pane should do the trick).
4. Now, you must learn to think in terms of joint frames vs global frames.
5. Every joint has a parent link (body) and a child link. Every joint has a parent joint.
6. The reference frame for the joint is the parent link's joint frame. The position and orientation of this joint must be represented with respect to this frame.
7. The reference frame for a child link is the joint of which it is a child.
8. Therefore, we must :
   (a) Note down the positions and orientations of the joints *with respect to their parent joints.*
   (b) In the files where we have the individual components separately, *align* the body such that the position with respect to the origin is the same as the position of the link with respect to its parent joint.
9. Let me give you an example. Here, the revolute joint at the top is at $(0, 0.05, 0.475)$. The child of this is that bar which is supposed to swing about the Z axis. Now, with respect to its parent joint, the bar should be at $(0, 0, 0.005)$.
10. Make these modifications and save the STL files somwhere in the cloud, so that you can view the STL files in Linux.

11. Here's the final model files you should export:

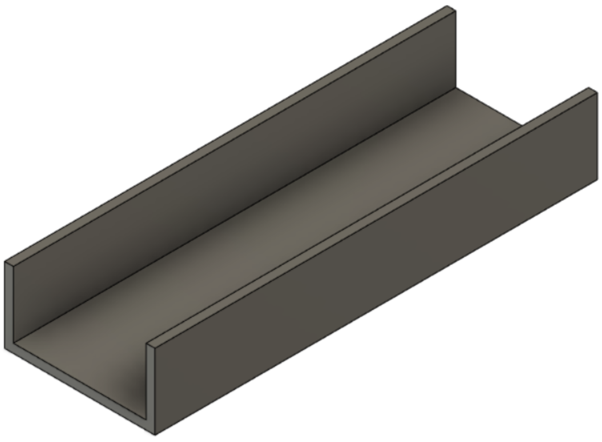


Figure 1: base.stl

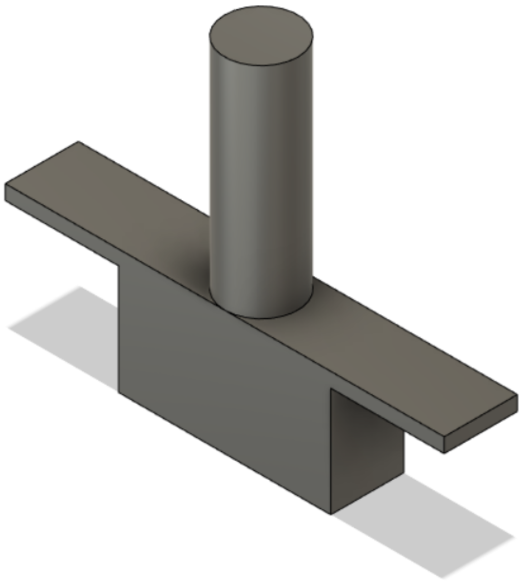


Figure 2: slider.stl

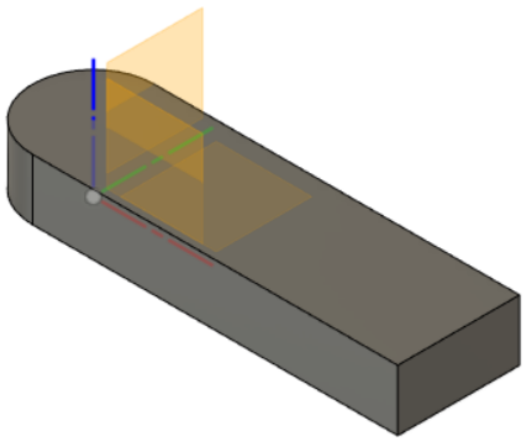


Figure 3: rotor.stl

3 Making the URDF file

3.1 A package for the robot

1. Create a package `turret_bot` which depends on `urdf`, `rviz` and `tf2`. You can choose to create a new workspace for this. For the sake of simplicity, I have not done this in the Github repo.

2. Create directories `urdf`, `meshes`, `launch`, `config` and `scripts` within the package directory.

3.2 The URDF file

1. The **Unified Robot Description Format** is an XML-based format for describing a robot model. Here, the links are considered to be nodes of a tree, with a clear parent-child relationship between links (this is a disadvantage for closed- chain robots).

2. Create `turret_bot/urdf/turret_bot.urdf`. The file:

```xml
<?xml version="1.0"?>
<robot name="turret_bot">
  <link name="base">
    <visual>
      <geometry>
        <mesh filename="package://turret_bot/meshes/base.stl" scale="1 1 1"/>
      </geometry>
      <material name="red">
        <color rgba="0.8 0 0 1"/>
      </material>
    </visual>
  </link>
  <link name="slider">
    <visual>
      <geometry>
        <mesh filename="package://turret_bot/meshes/slider.stl" scale="1 1 1"/>
      </geometry>
      <material name = "green">
        <color rgba="0 0.8 0 1"/>
      </material>
    </visual>
  </link>
  <link name="rotor">
    <visual>
      <geometry>
        <mesh filename="package://turret_bot/meshes/rotor.stl" scale="1 1 1"/>
      </geometry>
      <material name = "blue">
        <color rgba="0 0 0.8 1"/>
      </material>
    </visual>
  </link>
  <joint name="j1" type="prismatic">
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <axis xyz="0 1 0"/>
    <limit lower="0" upper="1" effort="1000" velocity="50"/>
    <parent link="base"/>
    <child link="slider"/>
  </joint>
  <joint name="j2" type="revolute">
    <origin xyz="0 0.05 0.475" rpy="0.0 0.0 0.0"/>
    <axis xyz="0 0 1"/>
    <limit lower="-3.141592" upper="3.141592" effort="1000" velocity="50"/>
    <parent link="slider"/>
    <child link="rotor"/>
  </joint>
</robot>
```

Breaking it down:

(a) `<robot name="turret_bot">` – all your links and joints should be between the `robot` tags.

(b) `<link name="base">` – describes a link. Since we will be referring to links later on, all links should be assigned a unique name.

(c) `<visual>` – defines the way the model looks. The `visual` properties are as opposed to `collision` or `inertial` properties of the robot. This tag does **NOT** define how the link behaves when in contact with other links, nor does it define the mechanical properties like mass or inertia.

(d) `<geometry>` – put all information about visual link geometry between these tags. We will reuse `geometry` when defining the collision properties later on. The geometry need not be a .STL mesh file – it can be one of many basic shapes too. For more information, see `http://wiki.ros.org/urdf/Tutorials` for more.

(e) `<mesh filename= ... scale="1 1 1"/>` – to import a .STL or .DAE file.

(f) `<material name= ...>` – to specify color, texture etc.

(g) `<color rgba="0.8 0 0 1"/>` – four floats between 0 and 1 to specify red, green, blue, and opacity.

(h) `<joint name= ... type= ...>` – give every joint a unique name. The type can be one of six types as mentioned at `http://wiki.ros.org/urdf/XML/joint`.

(i) `<origin xyz= ... rpy= ... />` specifies the position and orientation of the joint with respect to the parent joint (unless you have an `origin` tag in the parent link – in which case it may be more complicated. Experiment and see.). The roll-pitch-yaw system is described at the end of this document.

(j) `<axis xyz= .../>` – specify the direction of the joint axis, with respect to its own frame.

(k) `<limit lower=... upper=... effort=... velocity=.../>` – to specify the limits of the joint. `lower` and `upper` specify the position limits. **Note: Keep the limits non-overlapping i.e** $|upper - lower| < 2\pi$. Not doing this introduces a catastrophic bug in `gazebo_ros_control`'s position interfaces later on. `effort` specifies the maximum force/torque the joint can exert. `velocity` specifies the maximum linear/angular velocity that the joint can actuate at. All these quantities are in SI units.**The limit tag is compulsory for revolute and prismatic joints**.

(l) `<parent link=.../>` and `<child link=.../>` – to specify which links are joined by this joint. Note that a link can only have one parent link. This limits us to open-chain robots – closed-chain robots are not possible in the URDF file format.

4 RViz and the launch file

1. Create `urdf_rviz.launch` in the `launch` folder. The file:

```
<?xml version="1.0"?>
<launch>
    <param name = "robot_description"
        textfile = "$(find turret_bot)/urdf/turret_bot.urdf" />
    <node name="robot_state_publisher" pkg="robot_state_publisher"
         type="robot_state_publisher"/>
    <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
         type="joint_state_publisher_gui"/>
    <node name="rviz" pkg="rviz" type="rviz" required = "true"/>
</launch>
```

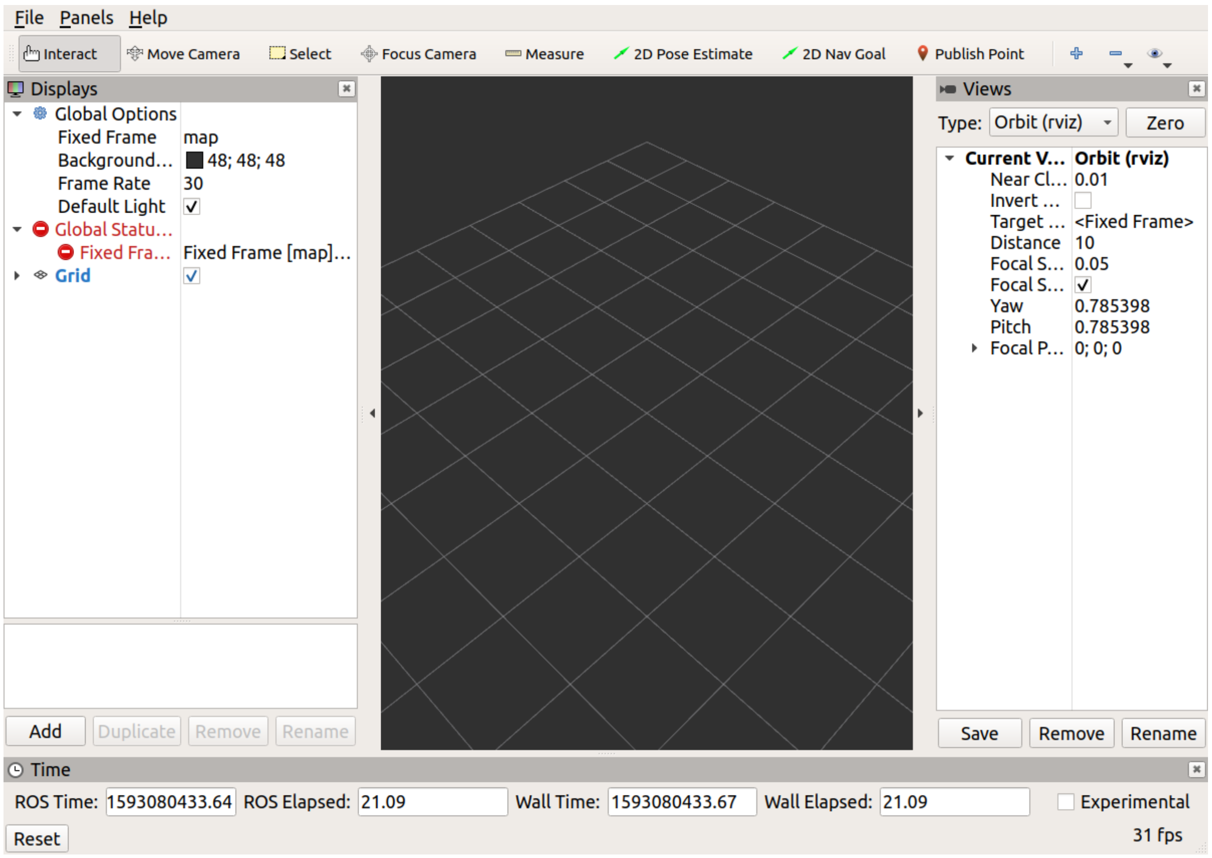
(a) `robot_description` is loaded to the parameter server. This is a common step in ROS.

(b) `robot_state_publisher` publishes the reference frame information to `tf2`, which is the back-end for RViz.

(c) `joint_state_publisher_gui` gives us a slider-based system to control the joint angles.

(d) `rviz` – RViz is a robot visualization utility provided along with ROS. It can be used to visualize the robot's geometry and to visualize the robot's sensor data in real-time.

2. Launch the launch file. You should see the following image.

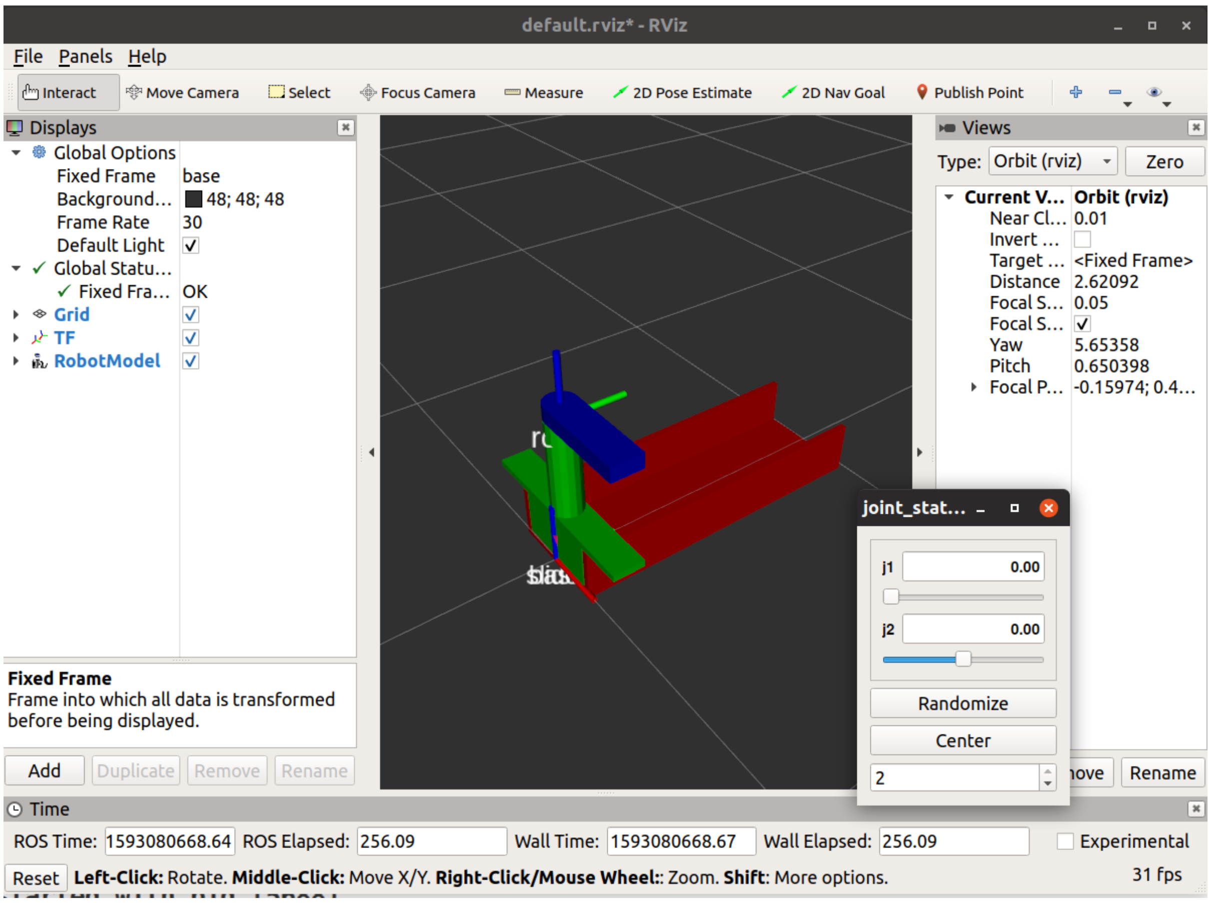


Global Options
Fixed Frame map
Background... 48; 48; 48
Frame Rate 30
Default Light ✓
Global Statu...
Fixed Fra... Fixed Frame [map]...
Grid ✓

Type: Orbit (rviz)
Current V... Orbit (rviz)
Near Cl... 0.01
Invert ...
Target ... <Fixed Frame>
Distance 10
Focal S... 0.05
Focal S... ✓
Yaw 0.785398
Pitch 0.785398
Focal P... 0; 0; 0

ROS Time: 1593080433.64 ROS Elapsed: 21.09 Wall Time: 1593080433.67 Wall Elapsed: 21.09 Experimental
31 fps

3. Double click *map* which is the field for *Fixed frame* and change it to `base`.
4. Add – Rviz – TF and Add – Rviz – RobotModel. You should see:



Global Options	
Fixed Frame	base
Background...	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Statu...	
✓ Fixed Fra...	OK
Grid	✓
TF	✓
RobotModel	✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)

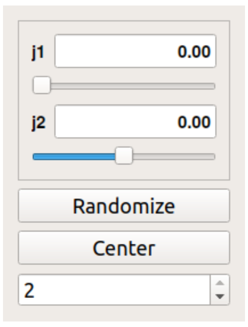
Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	☐
Target ...	<Fixed Frame>
Distance	2.62092
Focal S...	0.05
Focal S...	✓
Yaw	5.65358
Pitch	0.650398
Focal P...	-0.15974; 0.4...

joint_stat...
j1 0.00
j2 0.00
2

ROS Time: 1593080668.64 ROS Elapsed: 256.09 Wall Time: 1593080668.67 Wall Elapsed: 256.09

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

5. You should also see this:



j1 0.00
j2 0.00
2

You can control the sliders to modify the robot positions. Notice how the reference frames move along with the body.

5 On RPY fixed-frame angle systems

1. In the URDF, we saw an option `rpy =` in the attributes of `origin`. While we didn't have to use it, this feature is for specifying the orientation of the frame/link/joint. How it works is like this:

   (a) Start with the orientation of the parent frame.

   (b) Rotate about the center of your new frame about an axis parallel to the parent's X axis.

   (c) Rotate about the center of your new frame about an axis parallel to the parent's Y axis.

   (d) Rotate about the center of your new frame about an axis parallel to the parent's Z axis.

   Be sure to follow the same order. See `http://wiki.ros.org/urdf/XML/joint#Attributes` for more information.

6 Looking ahead

You have written a basic robot URDF file and can visualize the same in RViz. Next time, we will start simulating this same robot in Gazebo.