

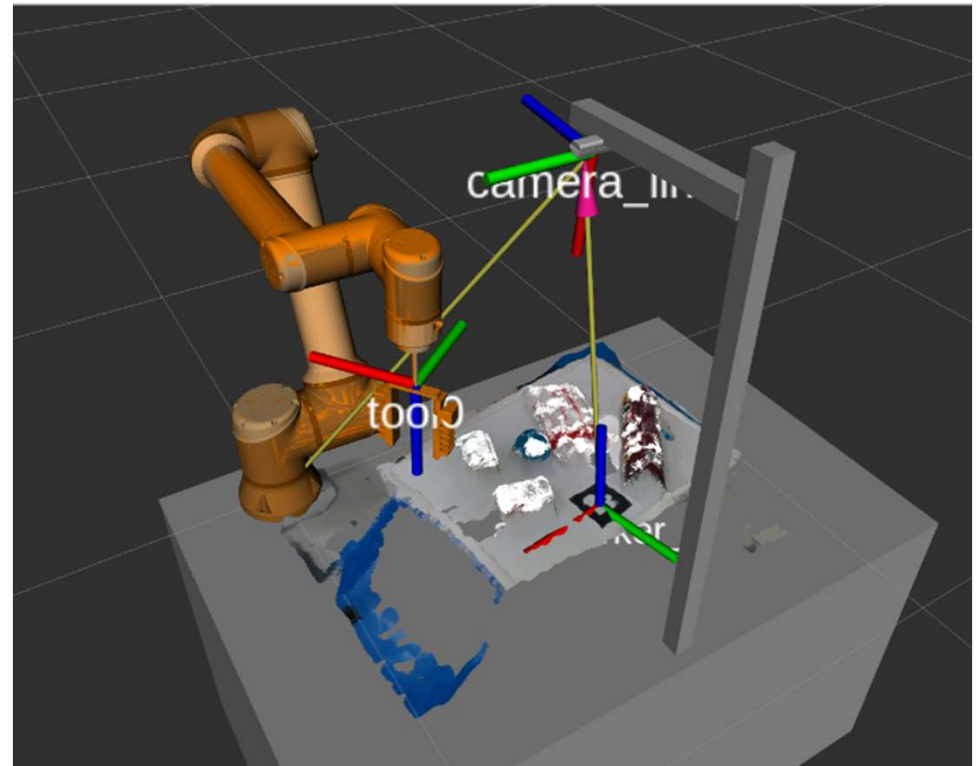


Module 9 ROS Hands-on Practice - URDF

Practice

- Designing a URDF for a robotic bin sorting application involves defining the structure, joints, and sensors of the robot.
- Robot: UR10e robot
- Gripper: Robotic 2F-85
- Camera: Realsense d435i Camera
- Fixture : Camera Stand (long bar, short bar)
Table
- All the CAD data are given, while regular shapes can be drawn

Layout



Download Mesh Files



git clone <https://github.com/xaviertse/mesh.git>

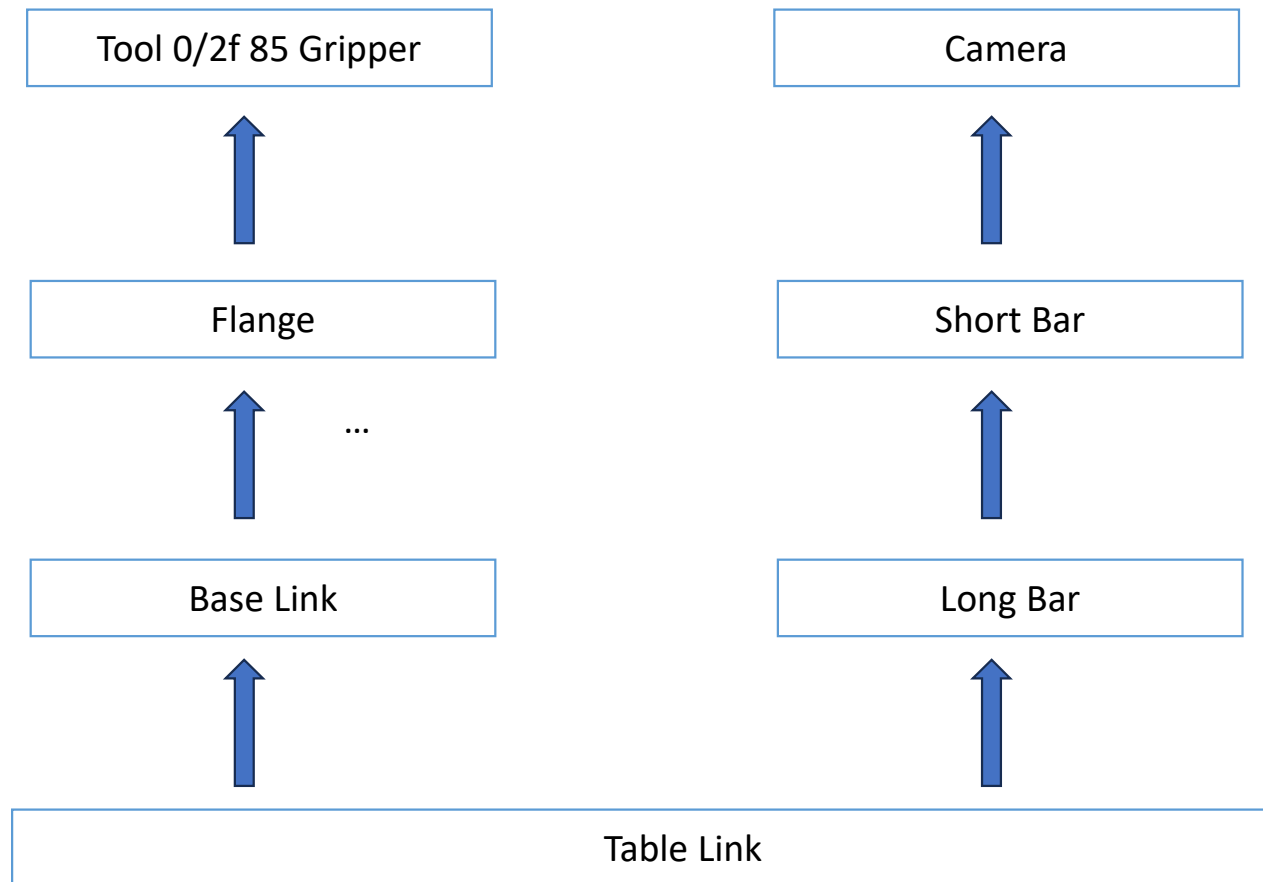


Download gripper CAD model 2f85.dae, RealSense Camera model d435.dae



Put both CAD data into Visual and Collision folder in the URDF folder

TF Tree



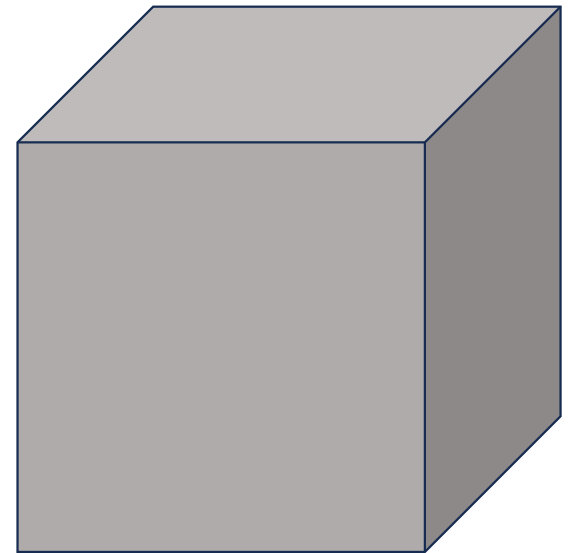
Step 1: Mount the Gripper on Robot as Tool 0

- Import mesh file 2f85.dae in Tool 0 Tag (Visual & Collision)
- Scale is 0.001
- Translation offset (0.09, 0, 0.01)



Step 2: Create a table

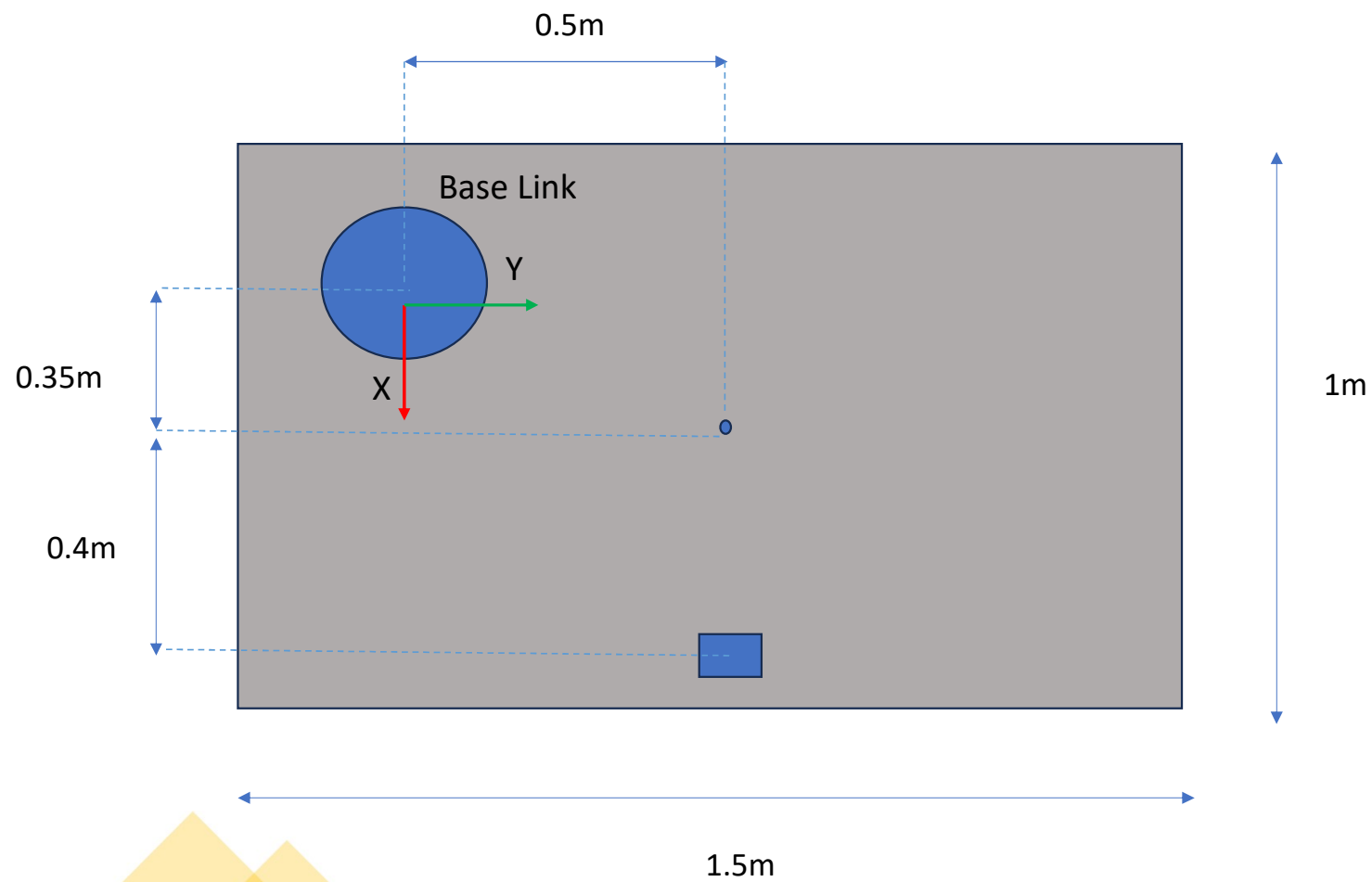
- Table size: 1m x 1.5m x 1m
- Color: Light Grey `rgba = (0.7 0.7 0.7 1.0)`
- Position: Robot Base mounted on tabletop surface



Create a Long Bar (Camera fixture)

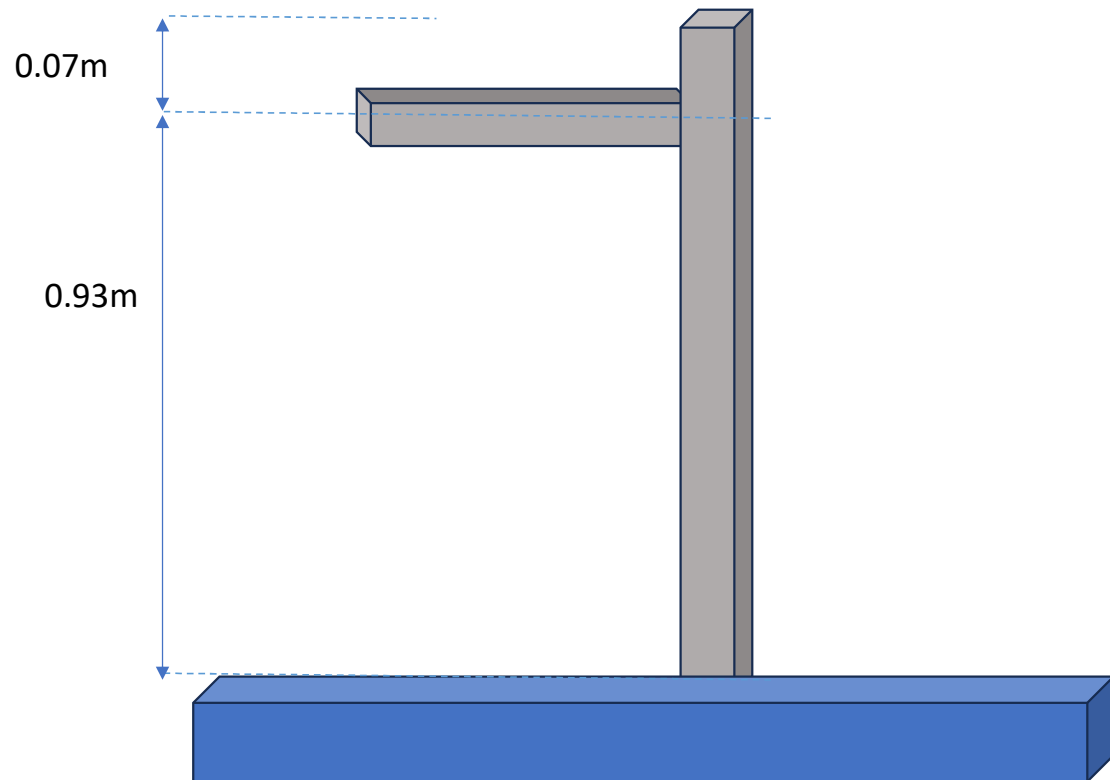
- Long Bar size: 0.04m x 0.04m x 1m
- Color: Light Grey rgba = (0.7 0.7 0.7 1.0)
- Position: Mounted on Tabletop Surface





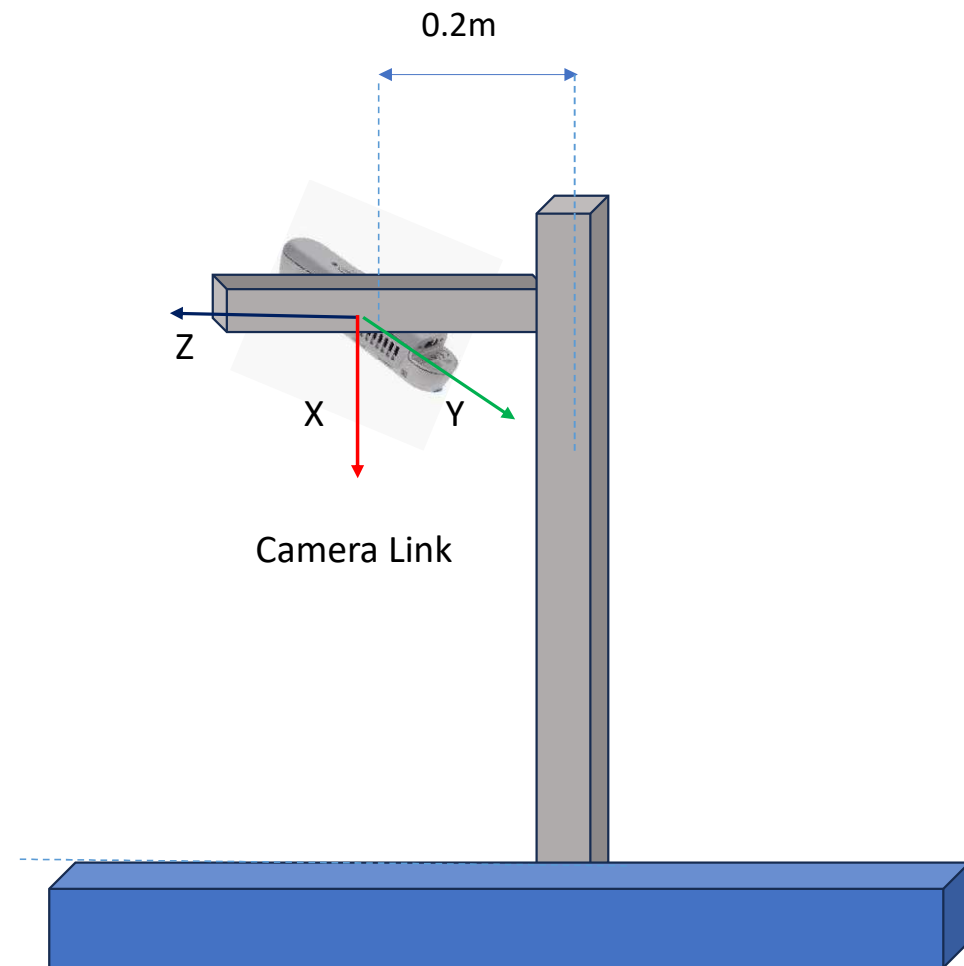
Create a Short Bar (Camera fixture)

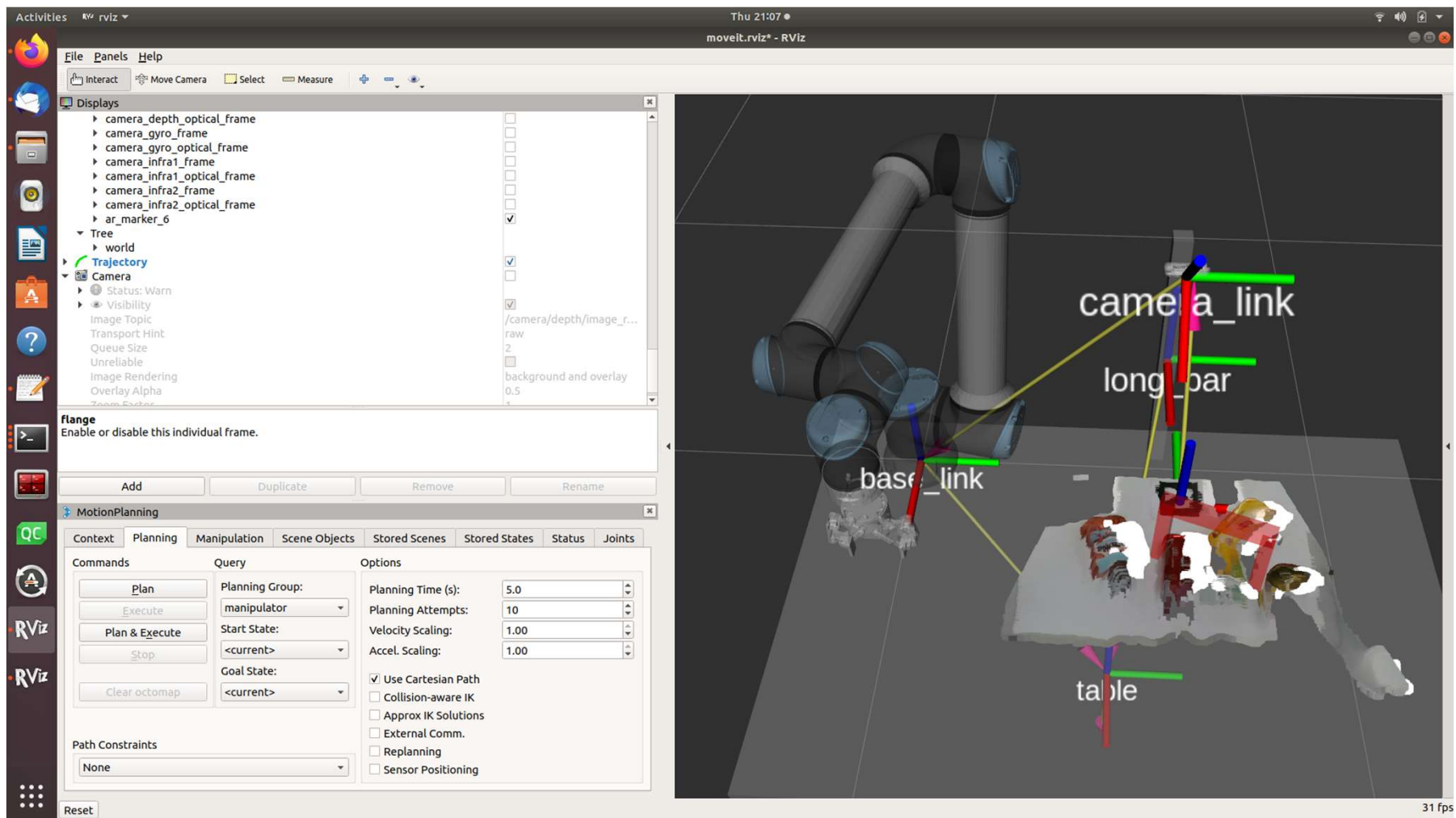
- Short Bar size: 0.04m x 0.04m x 0.4m
- Color: Light Grey `rgba = (0.7 0.7 0.7 1.0)`
- Position: Mounted on Long Bar with 90 degrees



Import Camera

- Import mesh file d435.dae in Tool 0 Tag (Visual & Collision)
- Scale is 1
- Camera 90 degrees pointing downwards



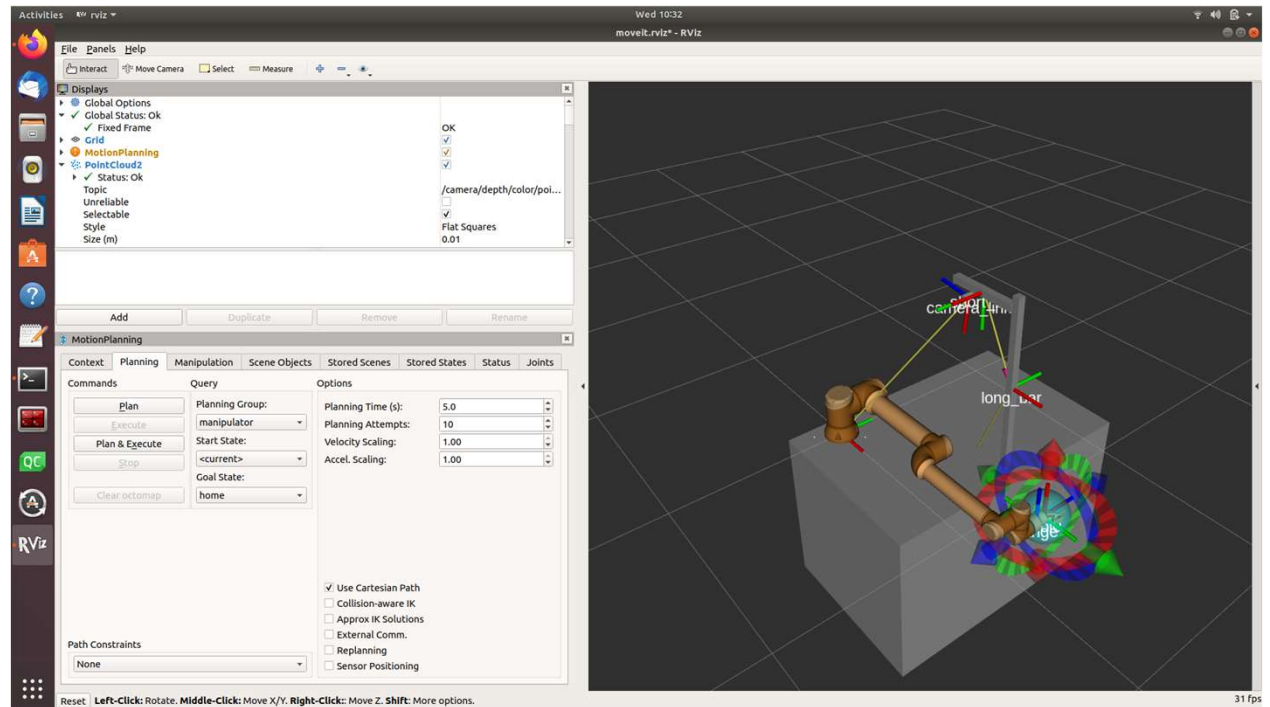


Disable Fixture Collison

- Go to SRDF file
- Type

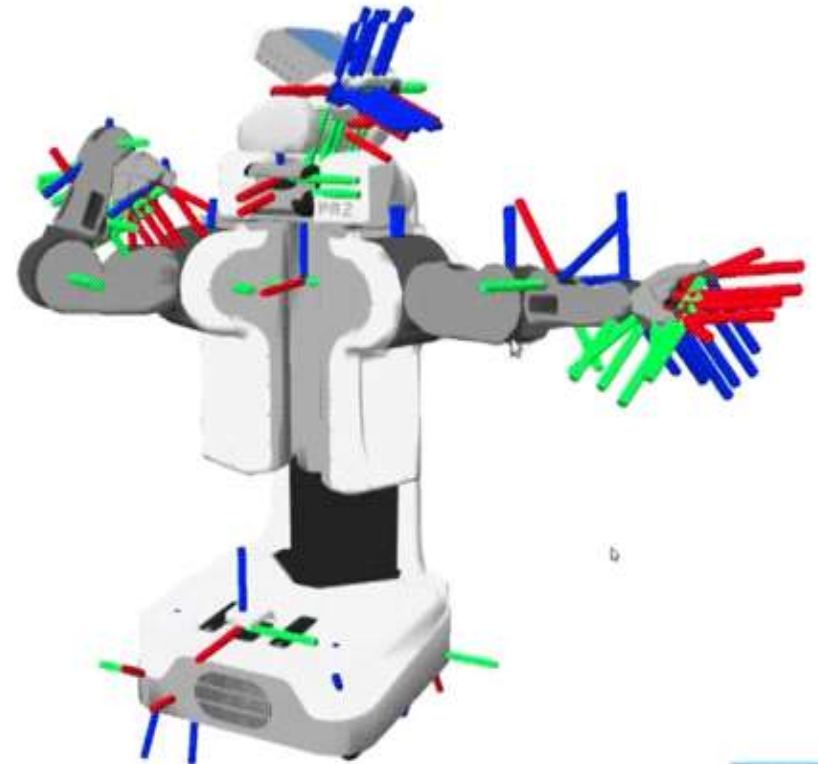
```
<disable_collisions link1="table" link2="base_link" reason="never" />  
<disable_collisions link1="table" link2="long_bar" reason="never" />  
<disable_collisions link1="short_bar" link2="camera_link" reason="never" />
```

Run UR Rviz Simulation

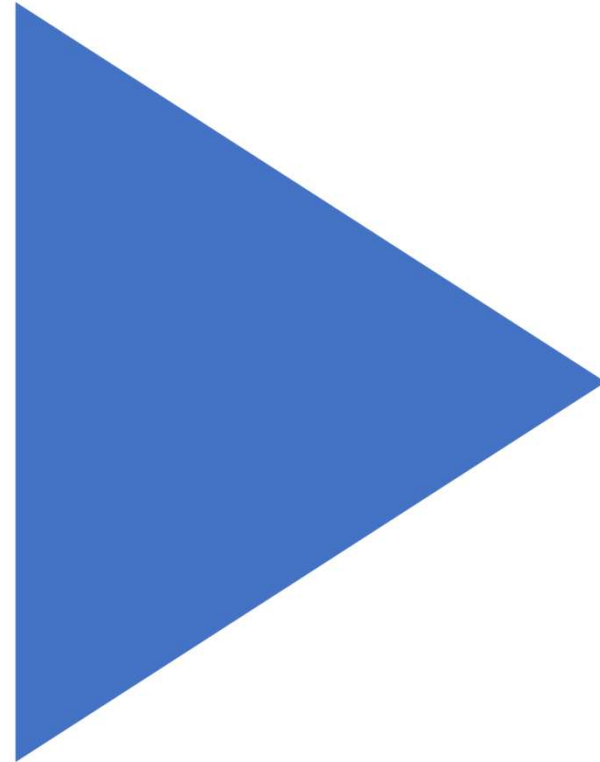


`roslaunch ur10e_moveit_config demo.launch`

Module 8: ROS Advanced



Previous
slides

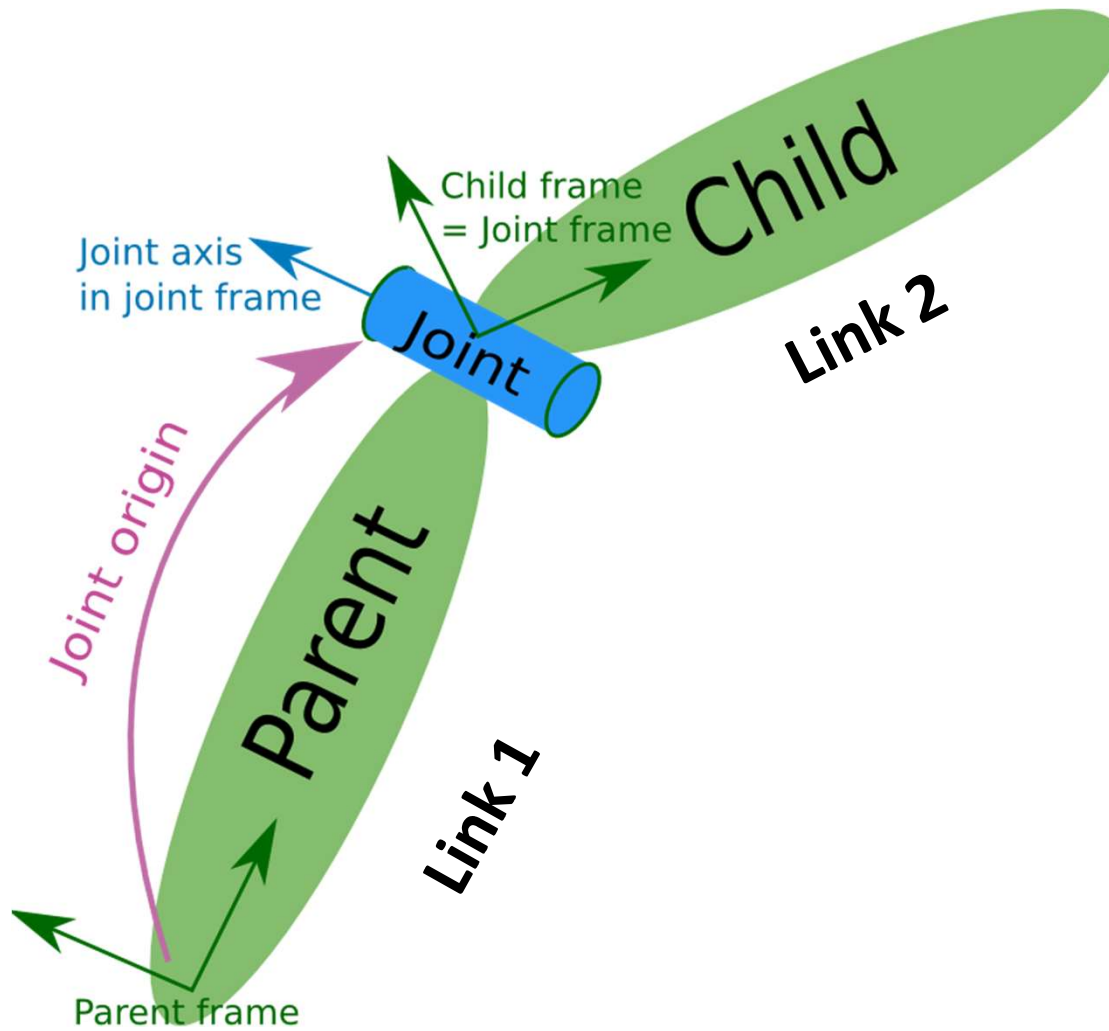


A large orange rounded rectangle with a white wavy line across its center.

URDF

URDF Example

```
1 <?xml version="1.0" ?>
2 <robot name="my_robot">
3   <link name="base_link">
4     <visual>
5       <!-- Visualization properties for the base link -->
6     </visual>
7     <collision>
8       <!-- Collision properties for the base link -->
9     </collision>
10  </link>
11
12  <joint name="joint1" type="revolute">
13    <origin xyz="0 0 0" rpy="0 0 0"/>
14    <parent link="base_link"/>
15    <child link="link1"/>
16    <axis xyz="0 0 1"/>
17    <limit lower="-3.1415" upper="3.1415" effort="100" velocity="0.1"/>
18  </joint>
19
20  <link name="link1">
21    <!-- Properties for link1 -->
22  </link>
23 </robot>
```



Links

The `<link>` element in a URDF file describes a rigid body or a link in a robot. It provides information about the physical properties of the link, such as its visual representation, collision properties, inertia, and more. Here's a breakdown of the key components within the `<link>` element:



A large orange shape on the left side of the slide, consisting of a rectangle with a quarter-circle cutout on its right side.

Link Name
(name
attribute)

- The name attribute uniquely identifies the link within the robot description.

```
<link name="base_link">
```

```
<!-- Link properties go here -->
```

```
</link>
```

A yellow dashed line in the bottom right corner of the slide, consisting of several short, curved segments.

Visual Properties (<visual>)

- The <visual> tag defines the visual representation of the link for visualization purposes. It includes geometry, material properties, and an origin.

```
<visual>
```

```
<geometry>
```

```
<!-- Define the visual geometry, e.g., box, cylinder, mesh -->
```

```
</geometry>
```

```
<material>
```

```
<!-- Define visual material properties like color or texture -->
```

```
</material>
```

```
<origin xyz="0 0 0" rpy="0 0 0"/>
```

```
</visual>
```



Visual Properties (`<visual>`)

Import CAD model

```
1 <link name="${prefix}tool0">
2   <visual>
3     <geometry>
4       <mesh filename="package://ur_description/meshes/ur10e/visual/Gripper.dae" scale="0.1 0.1 0.1"/>
5     </geometry>
6     <origin xyz="0 -0.5 0" rpy="0 pi ${-pi/2.0}"/>
7     <material name="LightGrey"/>
8   </visual>
9   <collision>
10    <geometry>
11      <mesh filename="package://ur_description/meshes/ur10/collision/Gripper.stl" scale="1 1 1"/>
12    </geometry>
13    <origin xyz="${0} ${0} ${0}" rpy="0 0 0"/>
14  </collision>
15 </link>
```

Supported CAD model format

1. COLLADA (COLLABorative Design Activity) (.dae):

COLLADA is a widely used interchange format for 3D assets. URDF can reference COLLADA files to describe the visual appearance of the robot's links.

2. STL (STereoLithography) (.stl): STL files are commonly used for representing 3D models composed of triangular facets. URDF can reference STL files for both visual and collision representations of the robot's links.

3. OBJ (Wavefront .obj): OBJ is a simple geometry format that defines the geometry of an object. URDF can reference OBJ files for visual representations.

4. Mesh formats: URDF can also support other mesh formats that are compatible with ROS and commonly used in 3D modeling software. This may include formats such as .mesh or .ply.

Collision Properties (`<collision>`)

- The `<collision>` tag describes the collision properties of the link, which are used in physics simulations to approximate how the robot interacts with its environment.

```
<collision>
```

```
<geometry>
```

```
<!-- Define the collision geometry, usually simpler  
than visual geometry -->
```

```
</geometry>
```

```
<!-- Collision-specific properties can be specified  
here -->
```

```
</collision>
```



Visual and Collision Geometry

- Inside the <geometry> tags of <visual> and <collision>, you can define the geometric shape of the link. This can be a box, cylinder, sphere, or even a mesh.

```
<geometry>
```

```
<box size="1 1 1"/>
```

```
</geometry>
```



Inertia (<inertial>)

- The <inertial> tag provides information about the link's inertia, which is essential for dynamics calculations.
- It includes the mass of the link and the rotational and translational inertia matrices.

```
<inertial>
```

```
<mass value="1.0"/>
```

```
<origin xyz="0 0 0" rpy="0 0 0"/>
```


```
<inertia ixx="0.1" ixy="0" ixz="0" iyy="0.1"  
iyz="0" izz="0.1"/>
```

```
</inertial>
```



Links example

```
1 <link name="base_link">
2   <visual>
3     <geometry>
4       <box size="1 1 1"/>
5     </geometry>
6     <material>
7       <color rgba="0.5 0.5 0.5 1"/>
8     </material>
9     <origin xyz="0 0 0" rpy="0 0 0"/>
10  </visual>
11  <collision>
12    <geometry>
13      <box size="1 1 1"/>
14    </geometry>
15  </collision>
16  <inertial>
17    <mass value="1.0"/>
18    <origin xyz="0 0 0" rpy="0 0 0"/>
19    <inertia ixx="0.1" ixy="0" ixz="0" iyy="0.1" iyz="0" izz="0.1"/>
20  </inertial>
21 </link>
```



Joints (<joint>)

The <joint> element in a URDF (Unified Robot Description Format) file describes the kinematic properties of a joint connecting two links in a robot. Joints define the relationship between links and allow for movement.

Here's an overview of the key components within the <joint> element:

Joint Name (name attribute)

- The name attribute uniquely identifies the joint within the robot description.

```
<joint name="joint1" type="revolute">
```

```
<!-- Joint properties go here -->
```

```
</joint>
```

Joint Type (type attribute)

The type attribute specifies the type of joint. Common types include:

- **revolute**: Rotational joint (e.g., a hinge joint), limited range specified by the upper and lower limits.
- **prismatic**: Translational joint (e.g., a sliding joint).
- **continuous**: Similar to revolute but with continuous rotation, no upper and lower limits

```
<joint name="joint1" type="revolute">
```

```
<!-- Other joint properties -->
```

```
</joint>
```

Joint Type (type attribute)

The type attribute specifies the type of joint. Common types include:

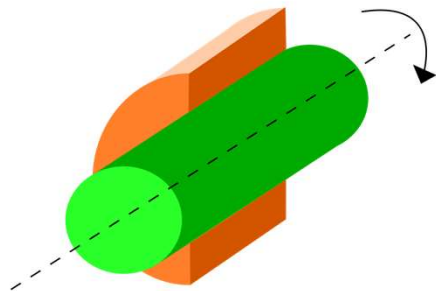
- **fixed** — this is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the <axis>, <calibration>, <dynamics>, <limits> or <safety_controller>.
- **floating** — this joint allows motion for all 6 degrees of freedom.
- **planar** — this joint allows motion in a plane perpendicular to the axis.

```
<joint name="joint1" type="planar">
```

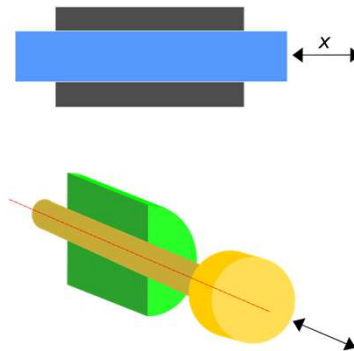
```
<!-- Other joint properties -->
```

```
</joint>
```

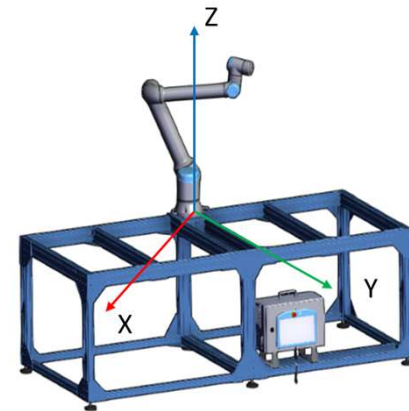

Joint Type (type attribute)




Revolute Joint



Prismatic Joint



Fixed Joint



Joint Origin (<origin> tag):

- The <origin> tag specifies the joint's position and orientation relative to its parent link.

```
<joint name="joint1" type="revolute">
```

```
<origin xyz="0 0 0" rpy="0 0 0"/>
```

```
<!-- Other joint properties -->
```


```
</joint>
```



Parent and Child Links (<parent> and <child> tags)

- The <parent> tag specifies the name of the parent link.
- The <child> tag specifies the name of the child link.

```
<joint name="joint1" type="revolute">  
  <parent link="parent_link"/>  
  <child link="child_link"/>  
  <!-- Other joint properties -->  
</joint>
```



Joint Axis (<axis> tag):


- The <axis> tag defines the joint axis. For example, in a revolute joint, this axis represents the axis of rotation.

```
<joint name="joint1" type="revolute">
```

```
<axis xyz="0 0 1"/>
```

```
<!-- Other joint properties -->
```

```
</joint>
```



Joint Limits (<limit> tag)

- The <limit> tag specifies constraints on joint motion, such as position limits, effort limits, and velocity limits.

```
<joint name="joint1" type="revolute">
```

```
<limit lower="-3.1415" upper="3.1415" effort="100" velocity="0.1"/>
```

```
<!-- Other joint properties -->
```

```
</joint>
```

Example

```
1 <joint name="joint1" type="revolute">  
2   <origin xyz="0 0 0" rpy="0 0 0"/>  
3   <parent link="parent_link"/>  
4   <child link="child_link"/>  
5   <axis xyz="0 0 1"/>  
6   <limit lower="-3.1415" upper="3.1415" effort="100" velocity="0.1"/>  
7 </joint>
```



XACRO

Xacro files typically have the ".xacro" extension and are preprocessed to generate standard XML files that can be used within ROS. This preprocessing step replaces macro invocations with their expanded definitions, allowing for more flexible and maintainable robot descriptions.

```

<?xml version="1.0"?>
<robot
  xmlns:xacro="http://wiki.ros.org/xacro">
    <xacro:include filename="$(find ur_description)/urdf/inc/ur_transmissions.xacro" />
    <xacro:include filename="$(find ur_description)/urdf/inc/ur_common.xacro" />
    <xacro:macro name="ur_robot" params="
prefix
joint_limits_parameters_file
kinematics_parameters_file
physical_parameters_file
visual_parameters_file
transmission_hw_interface:=hardware_interface/PositionJointInterface
safety_limits:=false
safety_pos_margin:=0.15
safety_k_position:=20"
>
      <!-- Load configuration data from the provided .yaml files -->
      <xacro:read_model_data
joint_limits_parameters_file="${joint_limits_parameters_file}"
kinematics_parameters_file="${kinematics_parameters_file}"
physical_parameters_file="${physical_parameters_file}"
visual_parameters_file="${visual_parameters_file}"/>
      <!-- Add URDF transmission elements (for ros_control) -->
      <xacro:ur_arm_transmission prefix="${prefix}" hw_interface="${transmission_hw_interface}" />
      <!-- links: main serial chain -->
      <link name="${prefix}base_link"/>
      <link name="${prefix}base_link_inertia">
        <visual>
          <origin xyz="0 0 0" rpy="0 0 ${pi}"/>
          <geometry>
            <mesh filename="${base_visual_mesh}"/>
          </geometry>
          <material name="${base_visual_material_name}">
            <color rgba="${base_visual_material_color}"/>
          </material>
        </link>
      </link>
    </macro>
  </robot>

```



```

    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 ${pi}"/>
        <geometry>
            <mesh filename="${base_collision_mesh}"/>
        </geometry>
    </collision>
    <xacro:cylinder_inertial radius="${base_inertia_radius}" length="${base_inert
        <origin xyz="0 0 0" rpy="0 0 0" />
    </xacro:cylinder_inertial>
</link>
<!-- joints: main serial chain -->
<joint name="${prefix}base_link-base_link_inertia" type="fixed">
    <parent link="${prefix}base_link" />
    <child link="${prefix}base_link_inertia" />
    <!-- 'base_link' is REP-103 aligned (so X+ forward), while the internal
frames of the robot/controller have X+ pointing backwards.
Use the joint between 'base_link' and 'base_link_inertia' (a dummy
link/frame) to introduce the necessary rotation over Z (of pi rad).
-->
        <origin xyz="0 0 0" rpy="0 0 ${pi}" />
    </joint>
    <joint name="${prefix}shoulder_pan_joint" type="revolute">
        <parent link="${prefix}base_link_inertia" />
        <child link="${prefix}shoulder_link" />
        <origin xyz="${shoulder_x} ${shoulder_y} ${shoulder_z}" rpy="${shoulder_roll}
        <axis xyz="0 0 1" />
        <limit lower="${shoulder_pan_lower_limit}" upper="${shoulder_pan_upper_limit}
effort="${shoulder_pan_effort_limit}" velocity="${shoulder_pan_velocity_limit}"/>
        <xacro:if value="${safety_limits}">
            <safety_controller soft_lower_limit="${shoulder_pan_lower_limit + saf
        </xacro:if>
        <dynamics damping="0" friction="0"/>
    </joint>
</xacro:macro>
</robot>

```