Building a Robot Simulation Environment in ROS

MSc. Robotics

Simon Bøgh

Associate Professor

Faculty of Engineering, Department of Materials and Production

Spring 2020

AALBORG UNIVERSITY
DENMARK

Content

- Introduction to URDF and XACRO
- Modifying Worlds
- Simple Geometry
- Using existing models

- ARIAC: Agile Robotics for Industrial Automation Competition
  - http://gazebosim.org/ariac

URDF

(Unified Robot Description Format)

Defining robot models

URDF overview

- Basics of URDF (Universal Robot Description Format)
- Links, joints, geometry
- Model environments and robots
- Understanding URDF
- Add and remove simple objects
- Import complex models
- Limitations
- Tools provided by ROS to work with URDF

URDF – What is it?

- Domain specific modeling language (DSML)
  - XML
- Stores:
  - Kinematics
  - Dynamics parameters and other meta-data
- Human and machine readable/writable
- ROS specific file-format based on XML
- Stores:
  - Robot body layout
  - Appearance
  - Extra information (joint position limits, joint velocity limits,..)
- Names and concepts from robotics domains

URDF - Implementation

- Text file containing XML text
- XML tags standardized in URDF "standard"
  - Can refer to other files e.g. 3D CAD models of robots
- File references using URLs
- 3D mesh files

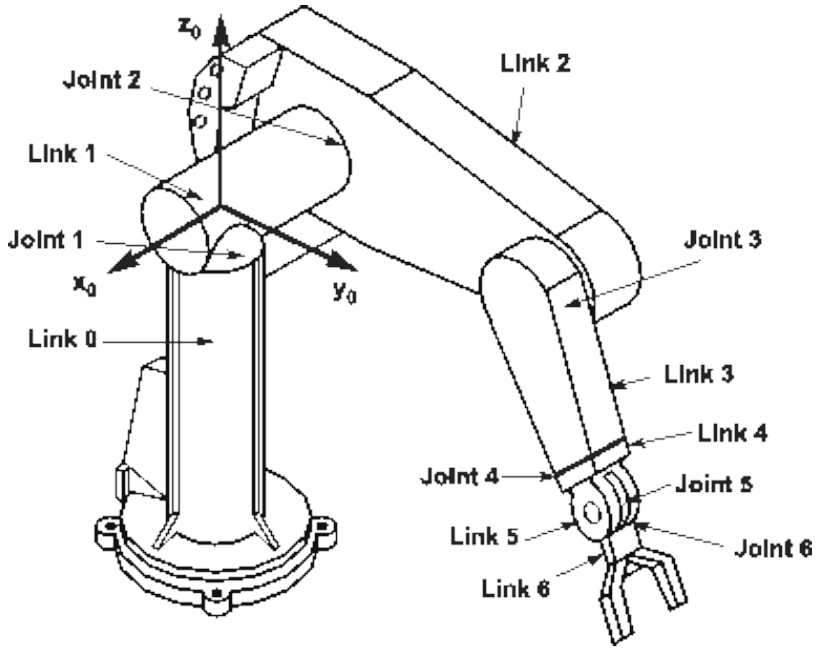
```
<?xml version="1.0"?>
<robot name="multipleshapes">

  <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>
```

URDF content

- Mostly `<link>` and `<joint>` elements
  - Corresponds to the links and joins of a robot
- `<link>`s
  - Form the robot's structure
- `<joint>`s
  - Form the connections between links and set motion constraints

URDF - Implementation

Joint and Link modeling



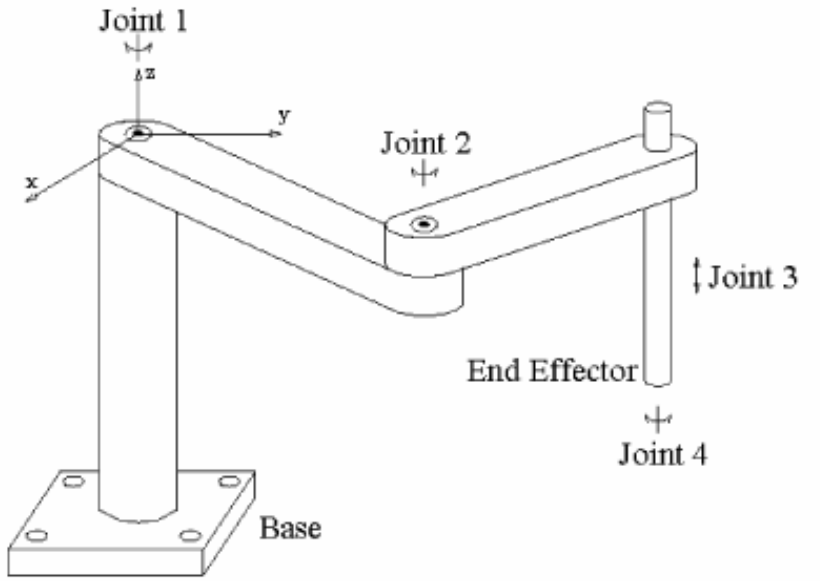


Figure 5. SCARA - Selective Compliance Assembly Robot Arm.

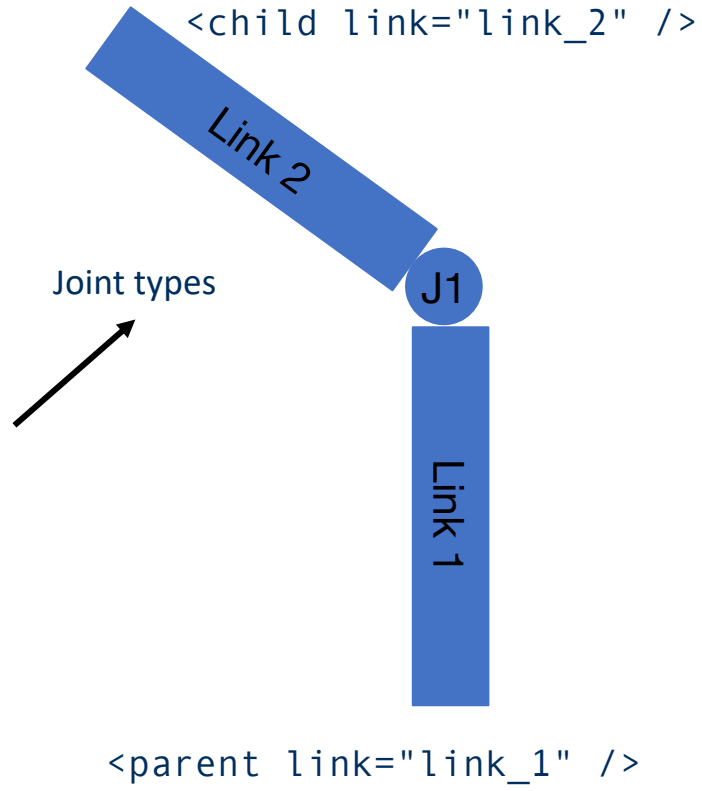
URDF - Example

tiny_robot.urdf

```
<robot name="tiny_robot">
   <link name="link_1" />
</robot>
```

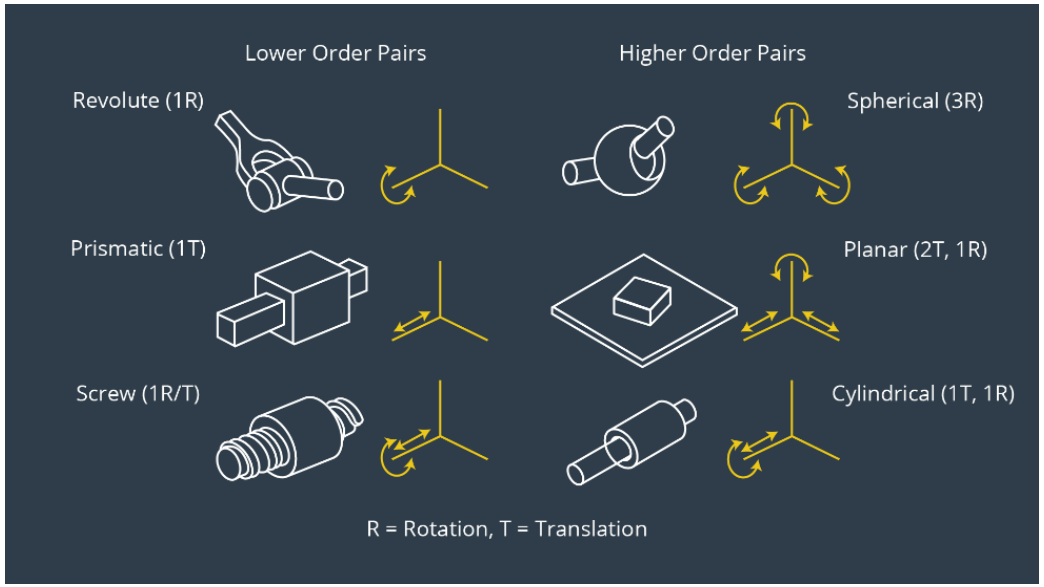
tiny_robot.urdf

```
<robot name="tiny_robot">
   <link name="link_1" />
   <link name="link_2" />
   <joint name="joint_1" type="..">
         <parent link="link_1" />
         <child link="link_2" />
   </joint>
</robot>
```



Joint types

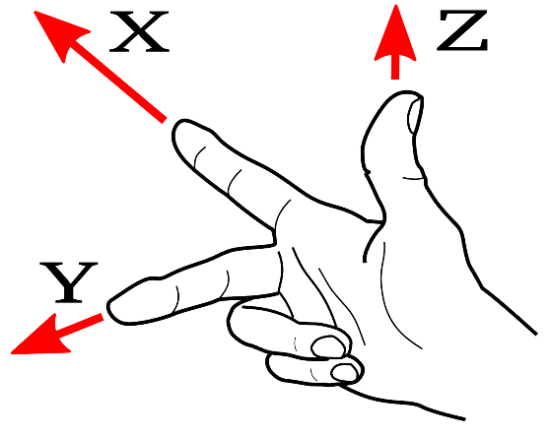
1. **Fixed**: rigid connection - This is not really a joint because it cannot move
2. **Revolute**: 1D rotation - a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits
3. **Continuous**: unlimited revolute - a continuous hinge joint that rotates around the axis and has no upper and lower limits
4. **Prismatic**: 1D translation - a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits
5. **Planar**: 2D translation - allows motion in a plane perpendicular to the axis
6. **Floating**: unlimited 6D - allows motion for all 6 degrees of freedom



https://jychstar.blogspot.com/2017/09/robot-nd-a2-kinematics.html

Standardisation

- REP (ROS Enhancement Proposal)
  - REP 103 - Standard Units of Measure and Coordinate Conventions [https://www.ros.org/reps/rep-0103.html]



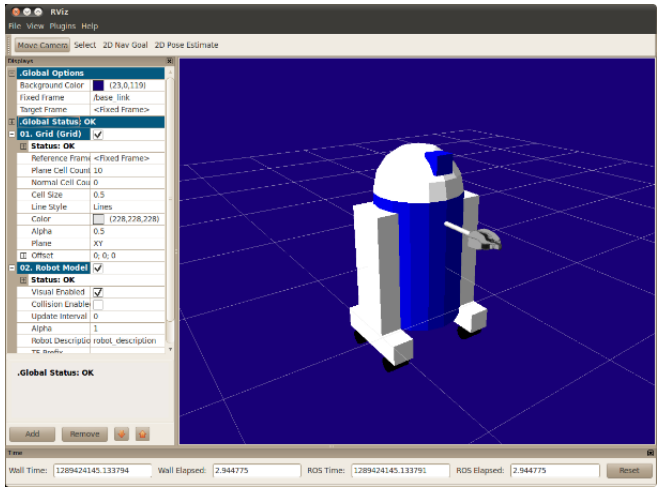
- ROS uses a right-handed coordinate system
  - X+ (forward) and Y+ (left)
  - → Z+ (up)
- ROS uses SI units:
  - Lengths: meters
  - Angles: radians (You can use functions to transform from/to radians)

URDF recap

- Text format based on XML
- Model robots with links and joints
- Various joint types are supported
- Links can be given using 3D mesh files for detailed appearances
  - **STL** is fairly common
  - Also supports **DAE** which can have its own color data, meaning you don't have to specify the color/material
    - <mesh filename="package://urdf_tutorial/meshes/my_cad_model.dae"/>
- ROS uses SI units for lengths (meters) and angles (radians)

URDF tutorial

- You'll follow and complete two of the tutorials about URDF and XACRO that are available on the ROS wiki.
  - Building a Visual Robot Model with URDF from Scratch
  - Building a Movable Robot Model with URDF
- Remember
  - The urdf_tutorial package has already been provided to you as part of the lecture 1 download zip file and should be part of your workspace
  - You do not need to install anything using apt-get or use rosdep
  - The files that are referred to in the tutorials are available in the urdf_tutorial/urdf directory.



URDF tools

- SolidWorks URDF exporter (tested in SW 2019)
  - http://wiki.ros.org/sw_urdf_exporter
  - http://wiki.ros.org/sw_urdf_exporter/Tutorials

- Simple online editor and visualizer
  - https://mymodelrobot.appspot.com/

Limitations of URDF

XACRO to the rescue

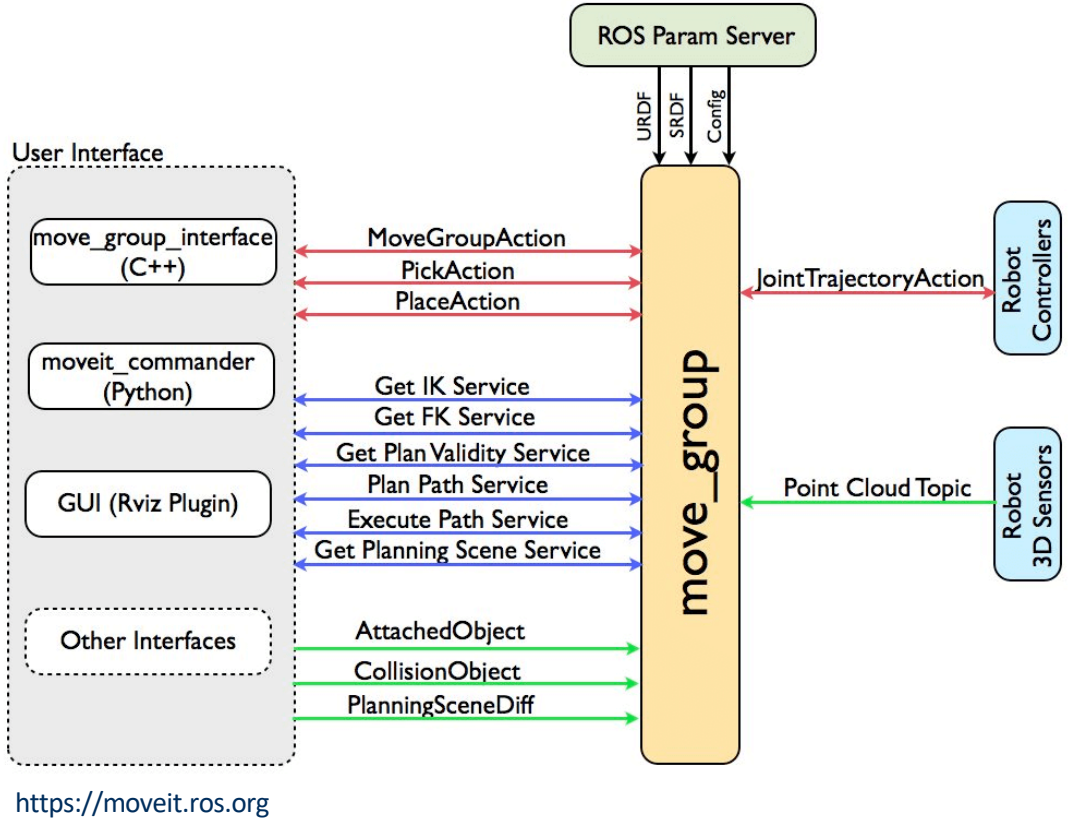
Limitations of URDF

- **Immutability**: Robot descriptions cannot be changed
- **No Loops**: Only tree structures
- **No sensor model**: No tags `<sensor></sensor>`
- **Composability**: Low reusability of URDFs

Limitation 1 – Immutability

- Result of typical control flow
  - Robot description read only once from parameter server
  - No standardized way to notify consumers of changes
  - Risk of desynchronization of nodes

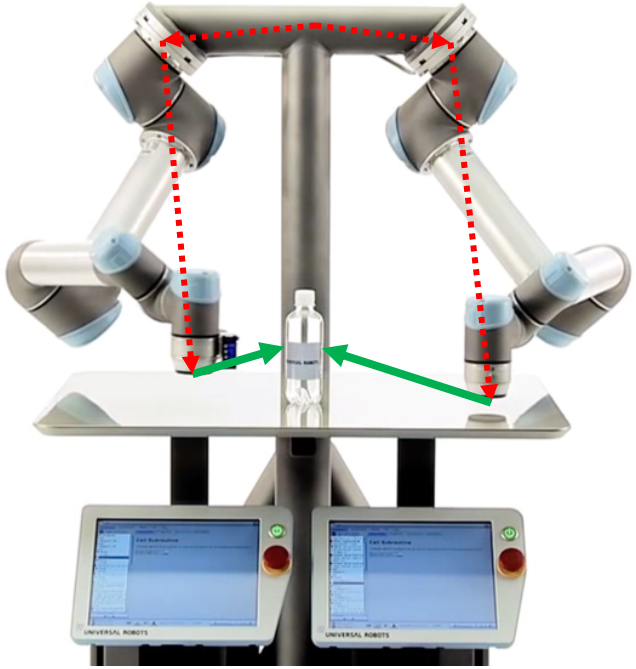
- **Examples where this is a problem**
  - Self-assembling robots
  - Robots with changing tools



https://moveit.ros.org

Limitation 2 – No loops/cycles

- Joints have only a single parent and child
- Only acyclic, directed graphs (trees) can be modeled
  - Real-world impact: **dual-arm manipulation**, parallel grippers, etc.
  - Robot torso, two arms, hands and the object, meaning that **the joints for the object need two parents**
  - This kind of setup is not supported and cannot be modeled with URDF



Limitation 3 – No sensor element

- Sensor meta-data cannot be incorporated into URDF directly
  - Like the resolution of the camera or the field-of-view
- Alternatives exist, but data is distributed
- However, if a URDF file is considered a description of a physical robot that any ROS application can use, then it is a delimitation
- It diminishes the value of URDF as central robot description

Limitation 4 – Reusability

- Only a single `<robot>` tag in a URDF
- No support for import of remote files
- Composite scenes have to be merged manually
- No way to compose multiple URDFs

Solution: XACRO (XML Macros)

- Programmatic URDF generation
- Create standalone URDF snippets: macros
- They can be added multiple times to a scene
- XACRO macros accept parameters

XACRO example

arm_macro.xacro

- Will be copied into the URDF automatically, and the parameters will be replaced by the values

```
<xacro:macro name="arm" params="parent arm_name">
   <link name="${arm_name}_link_1" />
   <joint name="${arm_name}_joint_1" type="..">
      <parent link="${parent}" />
      <child link="${arm_name}_link_1" />
   </joint>
</xacro:macro>
```

XACRO example

robot.xacro

- Easily add two arms to a robot by using arm.xacro

```
[..]
  <link name="torso" />
[..]
  <xacro:arm parent='torso' arm_name='left' />
  <xacro:arm parent='torso' arm_name='right' />
[..]
```

XACRO: Composability?

- Import macros from other files
- Parameterize templates
- Composite robots and scenes easier:
  - Import macro from file
  - Invoke macro

XACRO: Conversion

- Disadvantage: XACRO not directly compatible with URDF
- Transformation needed; we need to convert from XACRO to URDF
- Command:

```
$ rosrun xacro xacro /path/to/robot.xacro > robot.urdf
```

- Also checks for a valid XACRO file

Check URDF for syntax errors

- Validating URDF syntax: `check_urdf`
  - We first convert the XACRO file to URDF (this will generate the URDF file in the current working directory):

```
$ rosrun xacro xacro /path/to/robot.xacro > robot.urdf
```

  - And then run check_urdf on that file:

```
$ check_urdf robot.urdf
```

- This will check our URDF, but we must then map any mistakes it finds back to our XACRO files ourselves

Check URDF

tiny_robot.urdf (with error)

- Check the syntax (not semantics)
  - `$ check_urdf robot.urdf`

Result:

```
Error: Failed to build tree:
    child link [link_3] of joint [joint_1] not found
ERROR: Model Parsing the xml failed
```

```
<robot name="tiny_robot">
   <link name="link_1" />
   <link name="link_2" />
   <joint name="joint_1" type="continuous">
      <parent link="link_1" />
      <child link="link_3" />
   </joint>
</robot>
```

Check URDF

tiny_robot.urdf (corrected)

```
<robot name="tiny_robot">
   <link name="link_1" />
   <link name="link_2" />
   <joint name="joint_1" type="continuous">
      <parent link="link_1" />
      <child link="link_2" />
   </joint>
</robot>
```

Result:

```
robot name is: tiby_robot
---------- Successfully Parsed XML ----------
root Link: link_1 has 1 child(ren)
    child(1): link_2
```

Limitations and XACRO

- Limitations of URDF
  - Immutability
  - Only tree structures
  - No tags for sensors
  - Low reusability (composability)
- XACRO
  - Programmatic URDF generation
  - Macros, parameters, include other xacro files
- Check URDF syntax (first convert XACRO to URDF first)

```
$ check_urdf my_robot.urdf
$ urdf_to_graphiz my_robot.urdf  (generate PDF)
```

Cleaning up URDF with XACRO

```
<link name="base_link">
    <visual>
        <geometry>
            <cylinder length="0.6" radius="0.2"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder length="0.6" radius="0.2"/>
        </geometry>
    </collision>
</link>
```

```
<xacro:property name="width" value="0.2" />
<xacro:property name="bodylen" value="0.6" />
<link name="base_link">
    <visual>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
    </collision>
</link>
```

Math in XACRO: basic operations (+,-,*,/)

```
<cylinder radius="${wheeldiam/2}" length="0.1"/>
<origin xyz="${reflect*(width+.02)} 0 0.25" />

<link name="${5/6}"/> <!-- Evaluates to 0.833333333 -->
```

Include other XACRO files

```
<xacro:include filename="$(find my_package)/urdf/my_model/my_model.urdf.xacro"/>
<xacro:my_model_urdf my_model_parent="world_interface"/>
```

XACRO tricks

- **Common Trick 1**
  - Use a name prefix to get two similarly named objects
- **Common Trick 2**
  - Use math to calculate joint origins. In the case that you change the size of your robot, changing a property with some math to calculate the joint offset will save a lot of trouble
- **Common Trick 3**
  - Using a reflect parameter, and setting it to 1 or -1. For example, use the reflect parameter to put the legs on either side of the R2D2 body in the base_to_${prefix}_leg origin

Using Xacro to Clean Up a URDF File
http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File

Assignment

- With the first two URDF tutorials completed we can continue with the third tutorial now that we've discussed XACRO:
  - Using Xacro to Clean Up a URDF File
- Follow along with the instructions in the tutorial and see whether you understand what is being shown
- The tutorial refers to URDF snippets you've seen in the previous two tutorials ("Building a Visual Robot Model with URDF from Scratch" and "Building a Movable Robot Model with URDF"), so be sure to have those pages open for reference
- The files that are referred to in the tutorials are also available in the urdf_tutorial/urdf directory

Changing worlds

Modifying the URDF

Cleaning up our OMTP factory world

Cleaning up our OMTP factory world

- Go to “catkin_ws/src/lecture1/omtp_support/urdf/omtp_factory.xacro”
  - You can type roscd omtp_support/urdf/
  - if it doesn’t work remember to source the workspace first (source devel/setup.bash)
- Transform into urdf file

```
$ rosrun xacro xacro --inorder omtp_factory.xacro > omtp.urdf
```

- Check the syntax of the urdf file

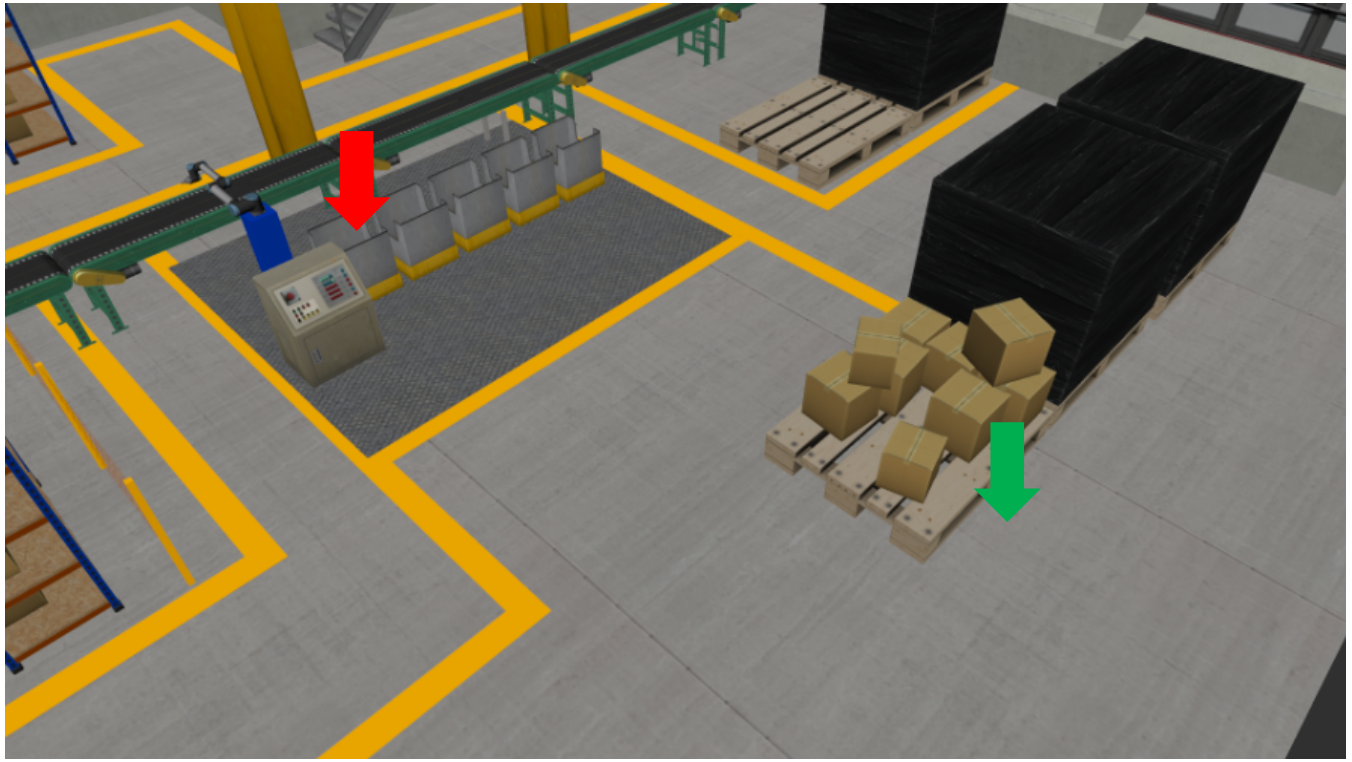
```
$ check_urdf omtp.urdf
```

- Visualize the factory in Rviz

```
$ roslaunch omtp_support visualize_omtp_factory.launch
```

Changing worlds

- Remove unneeded bins 2-5
- Move 1 bin to station 2 (green arrow)

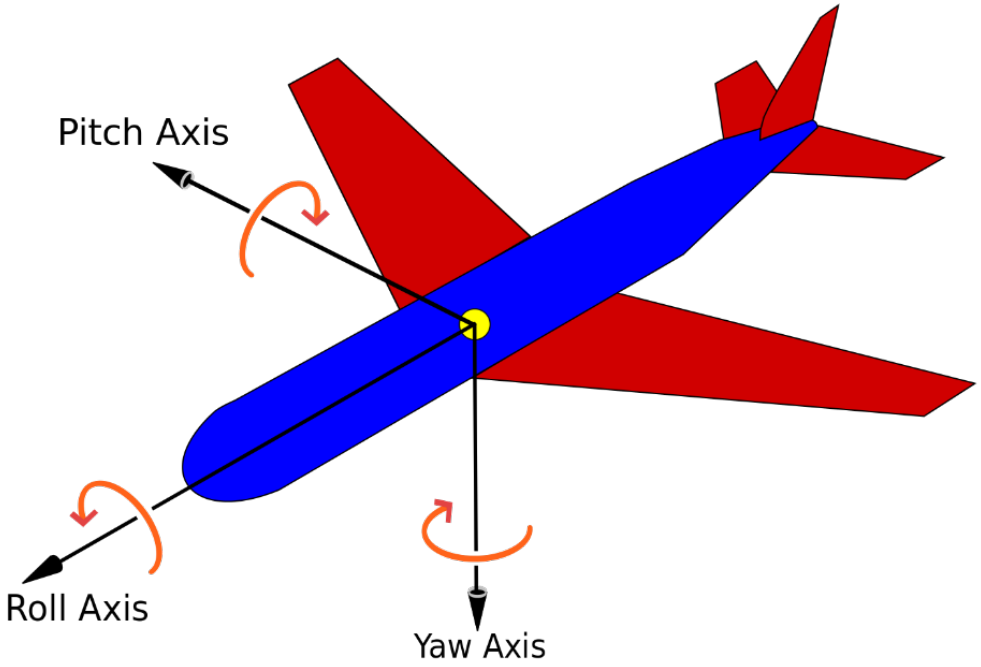


Changing worlds

- Joints: origin is the relative distance between the child and parent

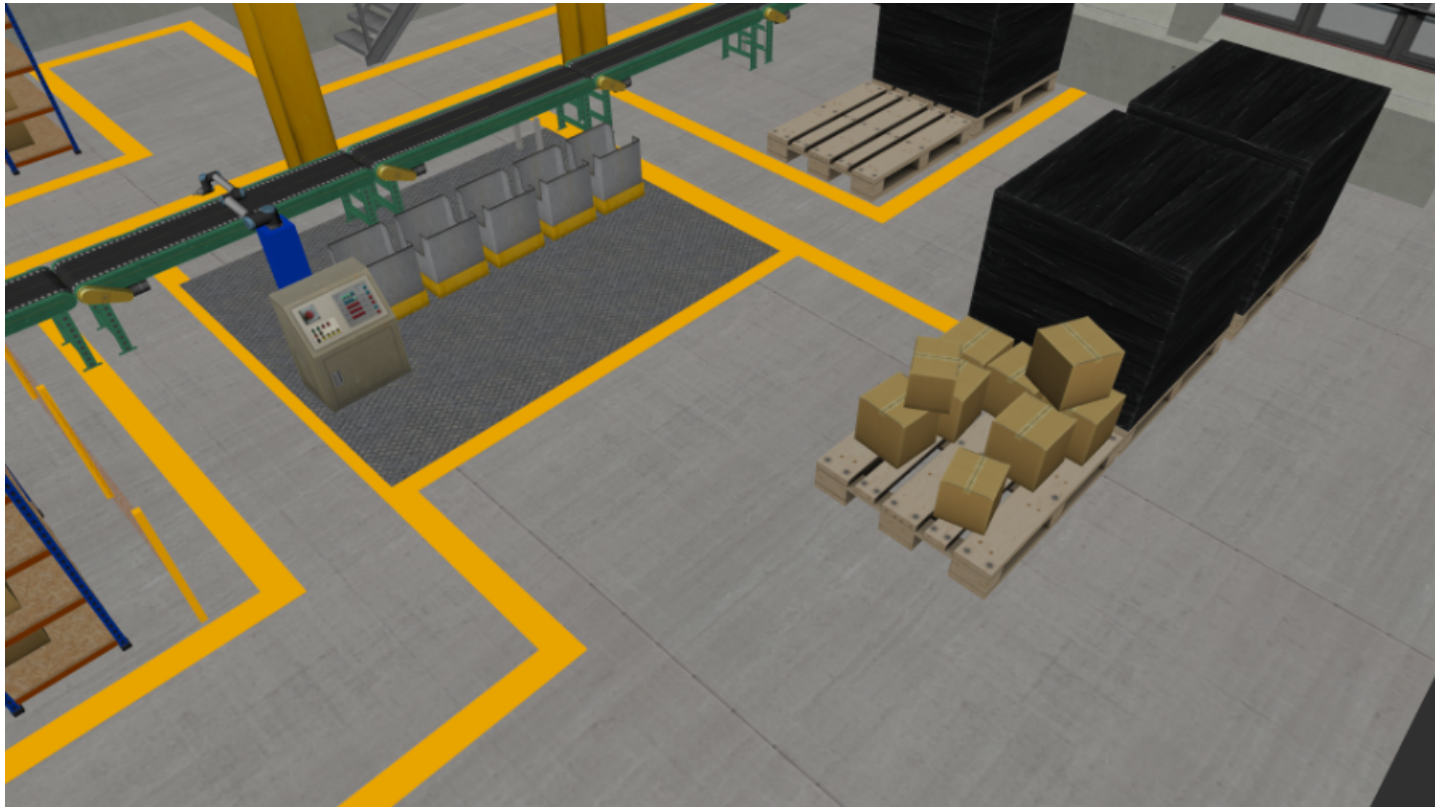
```
<!-- Joints -->
<joint name="world_interface_to_world" type="fixed">
  <parent link="world" />
  <child link="world_interface" />
</joint>
```

```
<!-- bin 1 -->
<joint name="bin_1_joint" type="fixed">
  <parent link="world_interface" />
  <child link="bin_1_base_link" />
  <origin xyz="-8.0 -2.2 0" rpy="0 0 0" />
</joint>
```



Changing worlds

- Add a robot pedestal
- Add a second or multiple robots



Using existing models

- Requires
  - Import the model definition (xacro macro)
  - Add the model to the factory (instantiation)
  - Fixing it in place (joint)
  - Updating orientation

- Add a UR5

```
<?xml version="1.0"?>
<xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro"/>
<xacro:ur5_robot prefix="robot2_" joint_limited="true"/>

<joint name="robot2-robot2_pedestal_joint" type="fixed">
   <parent link="robot2_pedestal_link"/>
   <child link="robot2_base_link"/>
   <origin xyz="0 0 0.6" rpy="0 0 ${radians(90)}"/>
</joint>
```

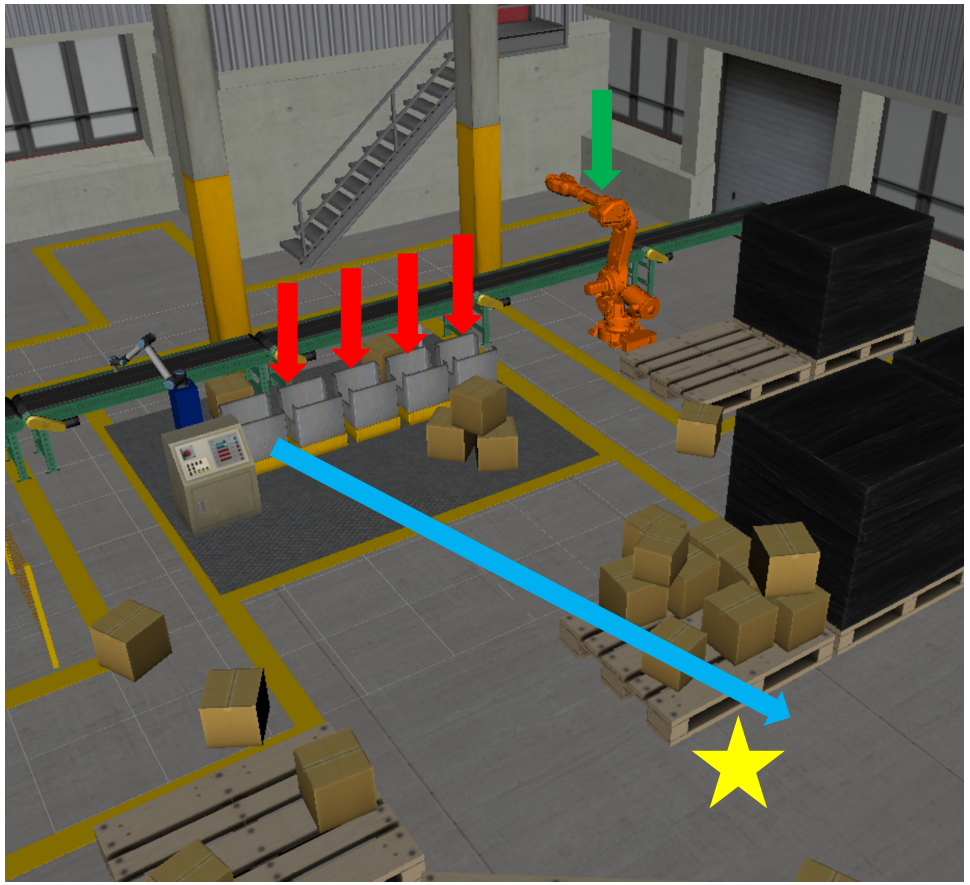
Additional Gazebo models

- More Gazebo models from OSRF
  - hg clone https://bitbucket.org/osrf/gazebo_models/
  - NOTE: hg (Mercurial) – alternative Version Control System to Git

Lecture 1 Assignments

1. URDF tutorials
2. XACRO tutorials
3. Clean up world XACRO, remove unneeded bins
4. Move bin 1 (blue arrow)
5. Add simple geometry
   - Adjust size and shape
   - Change origin
   - Update color (material)
   - Update location (update origin in the joint)
6. Add additional robots using an existing URDF/XACRO
   - Add another UR5
   - Add AAB IRB 6640 from ROS-Industrial

```
$ sudo apt install ros-melodic-abb
```



Submission requirements

- Code
  - Make a repository on Bitbucket for your team/group
  - The repository must include well-documented and organized code
  - Use tags to set versions for each lecture submission: `lec1_submission`
- README
  - Include `README.md` in the root of the repository
    - **Project Details**: The README describes the the project details including images/GIF/YouTube video
    - **Getting Started**: The README has instructions for installing dependencies or downloading needed files
    - **Instructions**: The README describes how to run the code in the repository. For additional resources on creating READMEs or using Markdown, ask Google ☺
    - **Ideas for Future Work**: The submission has concrete future ideas for improving the project
    - **Authors**: List of authors including emails