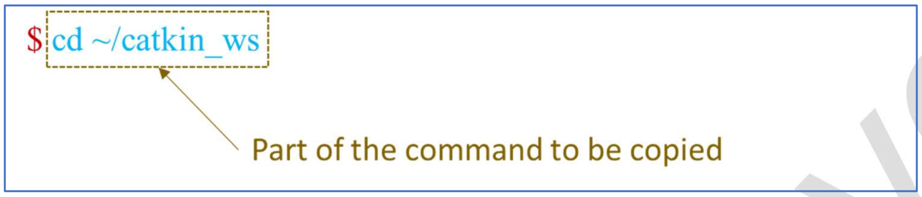
Importing Your ROS Package Generated from SolidWorks in ROS (ROBOT OPERATING SYSTEM)

Creation Date: 19-Sept-2022
Version of Ubuntu Used: 18.04 (Bionic) LTS
Version of ROS Used: ROS Melodic Morenia
Created By: Kunal B Aglave
Video Tutorial: https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12

1 Conventions Used

In this document, some colour coding is done to distinguish some notations. They are as given below

- **Command to execute in terminal:**
  To distinguish one command from another command, $ is used at the beginning of a new command. The part after the $ symbol is the actual command.

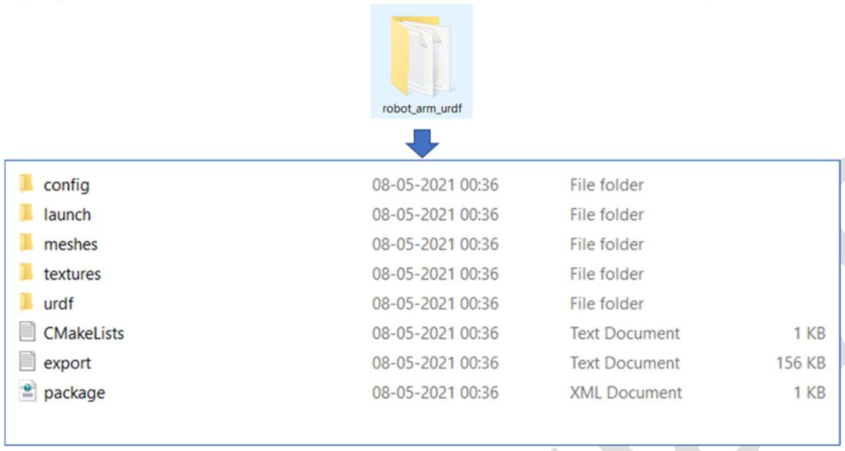


```
$cd ~/catkin_ws
```

  Reader should only copy the blue part of the command and not the $ symbol.
- **Red coloured Code:** The lines in a code snippet displayed by RED colour are already available in the file which is asked to be edited.
- **Green Coloured Code:** The lines in a code snippet displayed by GREEN colour need to be added by the reader in the file which is asked to be edited.
- **Purple Coloured Code:** The Part of the code snippet, reader need to change before pasting in the desired file.
- **Home directory:** “~/” this part in a command means your home directory.
- ***Notes:*** *Notes are written in italics.*
- **Comments:**
  - Comments in YAML starts with #
    #This is YAML comment
  - Comments in XML are enclosed in <!-- and -->
    <!--This is XML Comment -->

2 Overview of ROS Package Exported From SOLIDOWRKS

The ROS package generated from SOOLIDWORKS contains folders as given below.



config	08-05-2021 00:36	File folder	
launch	08-05-2021 00:36	File folder	
meshes	08-05-2021 00:36	File folder	
textures	08-05-2021 00:36	File folder	
urdf	08-05-2021 00:36	File folder	
CMakeLists	08-05-2021 00:36	Text Document	1 KB
export	08-05-2021 00:36	Text Document	156 KB
package	08-05-2021 00:36	XML Document	1 KB

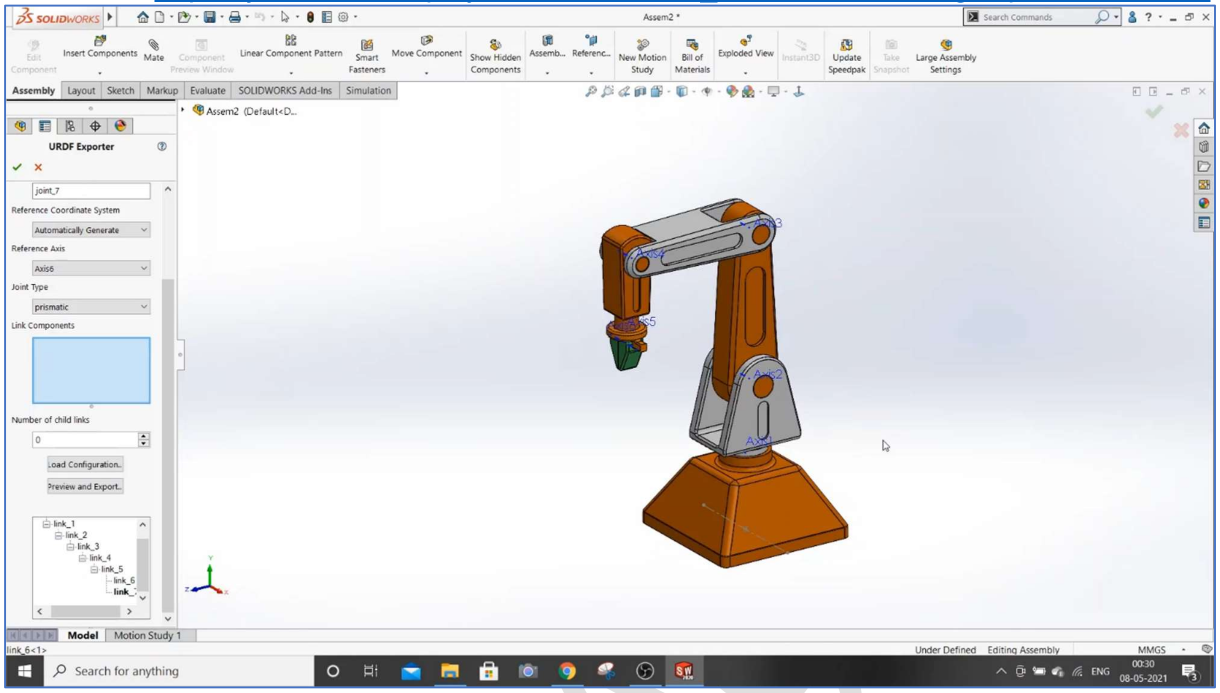
Folder and files available in the exported package:

- **config:** This folder contains a “joint_names_YourPackageName.yaml” file containing names of joints available in URDF file which is generated.
- **launch:** This folder contains two files. “display.launch” which is used to open your robot in Rviz. “gazebo.launch” is used to launch you robot in GAZEBO.
- **meshes:** This folder contains “.stl” files of all the links in your robot.
- **textures:** this folder will be empty.
- **urdf:** this folder contains a URDF file of your Robot. This file is a description of you robot containing information of links, joints, positions and ranges of joints, their mass properties, inertial properties etc. Also this folder contains a your_package_name.csv file containing information of your robots links.
- **CMakeLists.txt:** This file is the input to the CMake build system for building software packages [1]. It describes how to build the code and where to install it. You should not rename or change the sequence of code in this file.
- **export:** Contains logs generated while creating the package in SOLIDWORKS
- **package.xml:** This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages. Your system package dependencies are declared in package.xml. If they are missing or incorrect your package may or may not work. [2]

If you are familiar with ROS, you can directly copy the folder generated from soliworks directly to your Catkin Workspace. Once you copy it, build your Catkin Workspace and launch the gazeb.launch or display.launch file.

But the default generated package is just a robot description. If you don’t modify it, you cannot give any motion commands, control the joints or any other thing.
In this tutorial, we will see, how you can modify the package generated from SOLIDWORKS and how to **simulate it using ROS, Rviz, Gazebo and Moveit.**

In Last Tutorial, we created the URDF using SOLIDWORKS from this Robotic Arm Model: https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12


URDF Exporter

joint_7

Reference Coordinate System: Automatically Generate

Reference Axis: Axis6

Joint Type: prismatic

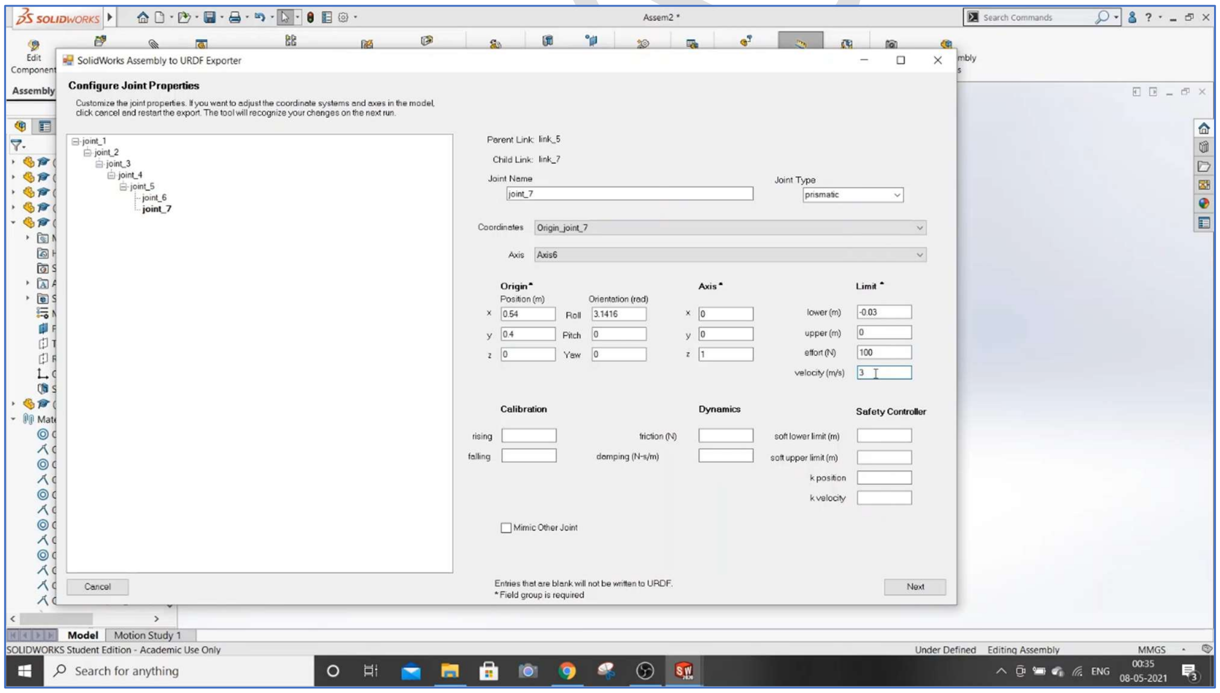
Link Components

Number of child links: 0

Load Configuration...

Preview and Export...

```
link_1
  link_2
    link_3
      link_4
        link_5
          link_6
          link_7
```


SolidWorks Assembly to URDF Exporter

Configure Joint Properties

Customize the joint properties. If you want to adjust the coordinate systems and axes in the model, click cancel and restart the export. The tool will recognize your changes on the next run.

```
joint_1
  joint_2
    joint_3
      joint_4
        joint_5
          joint_6
          joint_7
```

Parent Link: link_5

Child Link: link_7

Joint Name: joint_7

Joint Type: prismatic

Coordinates: Origin_joint_7

Axis: Axis6

Origin* Position (m)		Orientation (rad)		Axis*		Limit*	
x	0.54	Roll	3.1416	x	0	lower (m)	-0.03
y	0.4	Pitch	0	y	0	upper (m)	0
z	0	Yaw	0	z	1	effort (N)	100
						velocity (m/s)	3

Calibration		Dynamics		Safety Controller	
rising		friction (N)		soft lower limit (m)	
falling		damping (N-s/m)		soft upper limit (m)	
				k position	
				k velocity	

Mimic Other Joint

Entries that are blank will not be written to URDF.
* Field group is required

Cancel Next

3 Importing the package/URDF generated from SOLIDWORKS in ROS

Link to the full video tutorial: https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12

3.1 Copy the generated Package to Ubuntu

The first step is to take the package generated in your Ubuntu machine. Copy the package folder generated using SOLIDWORKS as it is to your Ubuntu machine having ROS Installed. Keep it at some suitable location, so it will be easily accessible. You can keep it on your desktop or any folder you want. I copied it using a pen drive. You can transfer using google drive as well.

3.2 ROS installation

If you haven't installed ROS yet, follow http://wiki.ros.org/melodic/Installation/Ubuntu this page.

I am using Ubuntu 18.04 (Bionic) and ROS Melodic Morenia.
Every ROS version is created for different Ubuntu versions. If you have another version of Ubuntu, download the version of ROS that is supported by your version of ubuntu. For more details: http://wiki.ros.org/Distributions

3.3 Setup a CATKIN Workspace

If you are using ROS from long time and have you catkin workspace already created, you can just use that for this tutorial.
But, if you are new or want to create a separate workspace for this project, just copy and run the commands given below in your Ubuntu Terminal one by one.

1. Update your Ubuntu packages
   ```
   $ sudo apt-get update
   ```
2. Go to home directory
   ```
   $ cd ~/
   ```
3. Create a catkin workspace folder along with src folder in it. I am creating a workspace with name "moveit_ws".
   ```
   $ mkdir --parents moveit_ws/src
   ```
   You can use any name for your workspace like "catkin_ws".
4. Go to the catkin workspace you created.
   ```
   $ cd moveit_ws
   ```
5. Initialize the Catkin workspace
   ```
   $ catkin init
   ```
6. Go to the catkin workspace directory that you created if not already in it.
   ```
   $ cd ~/moveit_ws
   ```
7. Build your workspace.
   ```
   $ catkin build
   ```
8. Source the "setup.bash" file automatically generated in your catkin workspace's "devel" folder
   If you are already in your catkin workspace:
   ```
   $ source devel/setup.bash
   ```
   If you are not in your catkin workspace, then give full path:
   ```
   $ source ~/moveit_ws/devel/setup.bash
   ```

3.4 Copy the ROS package folder you saved at suitable location to your catkin workspace.

Now, our workspace is ready. We need to copy our package that we created from SOIDWORKS in our catkin workspace.

Copy the ROS package created using SOLIDWORKS in the “src” folder of your catkin workspace.

Note: *The folder name generated from SOLIDWORKS is the name of the package. Do not change the name of folder. Because, the same name is used in the automatically generated scripts inside the package.*

My package name is robot_arm_urdf. I have copied in in following path:
~/moveit_ws/src/robot_arm_urdf

3.5 Edit the CmakeLists.txt

The CMakeLists.txt file is very basic and does not contain other important dependencies needed for our simulation. Edit the script in this file as given below

CmakeLists.txt [Before Editing]

```
cmake_minimum_required(VERSION 2.8.3)

project(robot_arm_urdf)

find_package(catkin REQUIRED)

catkin_package()

find_package(roslaunch)

foreach(dir config launch meshes urdf)
install(DIRECTORY ${dir}/
DESTINATION
${CATKIN_PACKAGE_SHARE_DESTINATION}/${dir})
endforeach(dir)
```

CmakeLists.txt [After Editing]

```
cmake_minimum_required(VERSION 2.8.3)

project(robot_arm_urdf)

find_package(catkin REQUIRED COMPONENTS
  message_generation
  roscpp
  rospy
  std_msgs
  geometry_msgs
  urdf
  xacro
  message_generation
)

catkin_package(
 CATKIN_DEPENDS
        geometry_msgs
        roscpp
        rospy
        std_msgs
)

find_package(roslaunch)

foreach(dir config launch meshes urdf)
            install(DIRECTORY ${dir}/
                        DESTINATION
${CATKIN_PACKAGE_SHARE_DESTINATION}/${dir})
endforeach(dir)
```

Note: Do not directly copy paste above code in your CMakeLists.txt file. Only add the lines displayed in green colour in the existing code which is represented by red colour.

3.6 Edit the package.xml file

The default package.xml has very few dependencies added. We need to add more dependencies for our simulation purpose.

package.xml [Before editing]

```
<package format="2">
  <name>robot_arm_urdf</name>
  <version>1.0.0</version>
  <description>
    <p>URDF Description package for robot_arm_urdf</p>
    <p>This package contains configuration data, 3D models and
launch files
for robot_arm_urdf robot</p>
  </description>
  <author>TODO</author>
  <maintainer email="TODO@email.com" />
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <depend>roslaunch</depend>
  <depend>robot_state_publisher</depend>
  <depend>rviz</depend>
  <depend>joint_state_publisher</depend>
  <depend>gazebo</depend>
  <export>
    <architecture_independent />
  </export>
</package>
```

package.xml [After editing]

```
<package format="2">
  <name>robot_arm_urdf</name>
  <version>1.0.0</version>
  <description>
    <p>URDF Description package for robot_arm_urdf</p>
    <p>This package contains configuration data, 3D models and launch files
for robot_arm_urdf robot</p>
  </description>
  <author>TODO</author>
  <maintainer email="TODO@gmail.com" />
  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>message_generation</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>urdf</build_depend>
  <build_depend>xacro</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>

  <depend>roslaunch</depend>
  <depend>robot_state_publisher</depend>
  <depend>rviz</depend>
  <depend>joint_state_publisher</depend>
 <depend>joint_state_publisher_gui</depend>
  <depend>gazebo</depend>
  <depend>moveit_simple_controller_manager</depend>

  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>geometry_msgs</build_export_depend>
  <build_export_depend>urdf</build_export_depend>
  <build_export_depend>xacro</build_export_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>urdf</exec_depend>
  <exec_depend>xacro</exec_depend>
  <exec_depend>message_runtime</exec_depend>

  <export>
    <architecture_independent />
  </export>
</package>
```

3.7 Edit the URDF file

The default urdf generated from SOLIWORKS only contains information about the links and their joint. We need to add some actuators, that will give some power and motion to the joints. Also, we need to add a gazebo controller that will control our robot joints. Also, other necessary information needs to be added.

To know more about urdf file: http://wiki.ros.org/urdf/XML

Note:

Every URDF starts with a tag defining its xml version and encoding type.
<?xml version="1.0" encoding="utf-8"?>

After that, there may be come comments defined with <!--Comment --> these tags.
For example:

Then there is a <robot>. The actual definition of your robot starts from here. The <robot> tag will have name attribute in it.
<robot name="robot_arm_urdf">

Next there are a tags defining a link. The link definition starts with <link name="link name"> and ends with </link>. It contains other tags between these two tags.

To know about <link> tag: http://wiki.ros.org/urdf/XML/link

Then there is a <joint name="name of joint" type="type of joint"> </joint> tag. This defines the joint between two links and the range of motion of the joint. This tag contains other tags.

To know more about <joint> tag : http://wiki.ros.org/urdf/XML/joint

The URDF Ends with a closing tag of the robot tag: </robot>

Open the URDF file available in your packag's urdf folder and start making changes in it as given below:

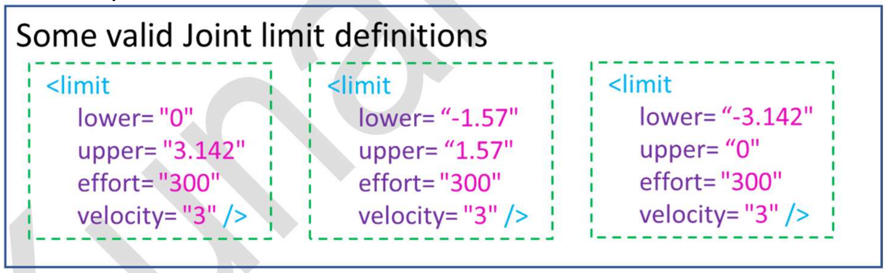
1. We need to add a "world" link and fix the "base_link" to it. Add below given code snippet in your urdf between the <robot name="robot_arm_urdf"> and the <link name="base_link"> tags as given below.

```
<robot
  name="robot_arm_urdf">

  <link name="world"/>
  <joint name="base_joint" type="fixed">
          <parent link="world"/>
          <child link="base_link"/>
          <origin rpy="0 0 0" xyz="0.0 0.0 0.17"/>
  </joint>

  <link
    name="base_link">
```

Note:

- *Your robot name will be same as the package name as you exported from SOLIDWROKS*
- *If you followed same instructions given by me in* https://youtu.be/I08lO_SRBbk *this video, then your base link name will be "base_link".*
- *If you used some other naming convention, just add the name of your base link (bottom most link) in* <child link= "Your_base_link_name"/> *tag.*

2. Check each <joint> tag in your URDF file. Make sure that, the "lower" limit of the joint is always lesser than the "upper" limit. In any case, if the "upper" limit is smaller than the lower "limit", just swap the position of these two.
   For example:

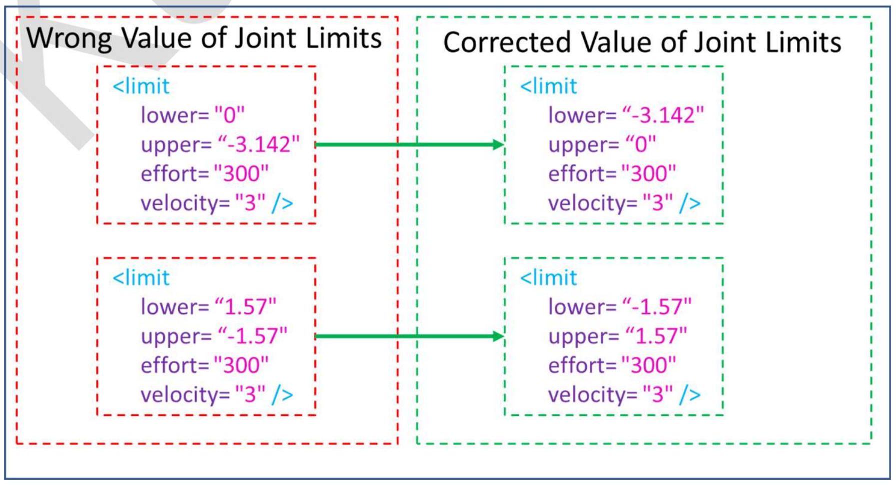


Some valid Joint limit definitions

```
<limit
   lower= "0"
   upper= "3.142"
   effort= "300"
   velocity= "3" />
```

```
<limit
   lower= "-1.57"
   upper= "1.57"
   effort= "300"
   velocity= "3" />
```

```
<limit
   lower= "-3.142"
   upper= "0"
   effort= "300"
   velocity= "3" />
```



Wrong Value of Joint Limits

```
<limit
   lower= "0"
   upper= "-3.142"
   effort= "300"
   velocity= "3" />
```

```
<limit
   lower= "1.57"
   upper= "-1.57"
   effort= "300"
   velocity= "3" />
```

Corrected Value of Joint Limits

```
<limit
   lower= "-3.142"
   upper= "0"
   effort= "300"
   velocity= "3" />
```

```
<limit
   lower= "-1.57"
   upper= "1.57"
   effort= "300"
   velocity= "3" />
```

3. Add transmission tags for each joint available in the URDF
   Copy the below code snippet and copy it at the end of the urdf before the </robot> tag. You need to add this code snippet for each joint in your URDF

```
<transmission name="link_n_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="joint_n">

        <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        </joint>
        <actuator name="link_n_motor">

        <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
                <mechanicalReduction>1</mechanicalReduction>
        </actuator>
</transmission>
```

   Note:
   - *Replace the "link_n" part with the name of child link of the joint and "joint_n" with the exact joint name for which you are writing this transmission tag.*
   - *The transmission and actuator names can be defined anything you want.*
   - *But, the joint name must be the exact name of the joint.*
   - *I have named my joints as "joint_1","joint_2"…."joint_n" and links as "link_1", "link_2"…."link_n". (Only the base link is named as "base_link". You should also do the same for base link).*
   - *My urdf has 7 joints. So, I just copied and pasted above snippet one after other and just changed the n value in each snippet as given in the video.*
4. Add Gazebo Controller
   After the last transmission tag, add a gazebo controller tag as given below.

```
 <gazebo>
        <plugin name="control"
filename="libgazebo_ros_control.so">
                <robotNamespace>/</robotNamespace>
        </plugin>
 </gazebo>
```

   Note:
   - *If you write controllers in the joint trajectory controller yaml file inside a namespace, then you need to change the "/" between the <robotNamespace> tags with "/yourNamespace".*
5. Add Self Collision tags.
   Similar to transmission tags, create the self collision tags for each movable link in urdf. For my case, "link_1" to "link_7" are movable. So, I will do it 7 times.

```
<gazebo reference="link_n">
  <selfCollide>true</selfCollide>
</gazebo>
```

   Note: *Change "link_n" with the name of the link.*

3.8 Write a .yaml file to define ROS controllers to be used for different joints and publishing joint states.

You need to create a .yaml file in the "config" folder of your ROS package. This file will contain a joint controller for: robotic arm joints, end effector joints, publishing the joint states and any other controller as needed.

To know more: http://wiki.ros.org/ros_control

In my case, joint_1 to joint_5 are of the robotic arm. Join_6 and joint_7 are end effector joints.

You can check this video to know more about my robotic arm: https://youtu.be/I08lO_SRBbk

Follow below steps to create a (.yaml) file containing ROS controller definition for our robot arm:

1. Open a terminal.
2. Go to the config folder of your robot arms package.
   ```
   $ cd ~/YourWorkSpaceName/src/YourUrdfPackageName/config
   ```
   For my case, it is:
   ```
   $ cd ~/moveit_ws/src/robot_arm_urdf/config
   ```
   If you have same workspace and package name just use above command.
3. Create a "joint_trajectory_controller.yaml" file in config folder of your urdf package using terminal.
   ```
   $ touch  joint_trajectory_controller.yaml
   ```
   Note: You can give any name to this file. You should use same file name while loading the controllers in the launch file.
4. Open the "joint_trajectory_controller.yaml" file in "gedit" text editor.
   ```
   $ gedit  joint_trajectory_controller.yaml
   ```
5. Copy and paste above code in it as it it.

```
#Instead of using TAB for indentation, use two spaces at the place of one TAB

#Controller to control robot arm joints
robot_arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_1, joint_2, joint_3, joint_4, joint_5]

#Controller to control end effector joints
hand_ee_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_6, joint_7]

#Controller to continuously publish joint states/positions
joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50
```

Note:

- *One can change the name for the controllers as they want. For ex. One can name "robot_arm_controller" like "my_arm_controller" or "yourArmName_arm_controller" etc.*
- *But, while spawning the controllers in launch file, you need to use the exact names there.*
- *Also, one need to add other joints of the arm or end effector in their respective controller separated by comma.*
- *One can use different "type" of controller as per their needs.*
- *Also, maintain proper indentation. Tabs are not accepted. Use uniform spacing for a level. One can use two spaces as one tab.*

6. Save and close the file.

3.9 Create a .launch file to spawn/open your robotic arm in gazebo

Launch file is used to execute multiple files/scripts/commands from a single file. Instead of launching each file needed for your robot simulation, launch file can be used to launch them all using a single launch file in a single terminal.

Follow below steps to create a launch file for our robot arm:

1. Open a terminal.
2. Go to the launch folder of your robot arms package.
   ```
   $ cd ~/YourWorkSpaceName/src/YourUrdfPackageName/launch
   ```
   For my case, it is:
   ```
   $ cd ~/moveit_ws/src/robot_arm_urdf/launch
   ```
   If you have same workspace and package name just use above command.
3. Create a “arm_urdf.launch” file in launch folder of your urdf package using terminal.
   ```
   $ touch  arm_urdf.launch
   ```
   Note: You can give any name to this file.
4. Open the “arm_urdf.launch” file in “gedit” text editor.
   ```
   $ gedit  arm_urdf.launch
   ```
5. Copy and paste the below code in the file.

```
<launch>
        <arg name="arg_x" default="0.00" />
        <arg name="arg_y" default="0.00" />
        <arg name="arg_z" default="0.00" />
        <arg name="arg_R" default="0.00" />
        <arg name="arg_P" default="0.00" />
        <arg name="arg_Y" default="0.00" />

        <!--Urdf file path-->
        <param name="robot_description" textfile="$(find Your_package_name)/urdf/urdf_file_name.urdf"/>

        <!--spawn a empty gazebo world-->
        <include file="$(find gazebo_ros)/launch/empty_world.launch" />
        <node name="tf_footprint_base" pkg="tf" type="static_transform_publisher" args="0 0 0 0 0 0
base_link base_footprint 40" />

        <!--spawn model-->
        <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-x $(arg arg_x) -y $(arg
arg_y) -z $(arg arg_z) -Y $(arg arg_Y) -param robot_description -urdf -model
Your_Robot_name_defined_in_urdf_file -J joint_1 0.0 -J joint_2 0.0 -J joint_3 0.0 -J joint_4 0.0 -J joint_5 0.0 -
J joint_6 0.0 -J joint_7 0.0" />

        <!--Load and launch the joint trajectory controller-->
        <rosparam file ="$(find Your_package_name)/config/Your_arm_trajectory_contoller_file_name.yaml"
command="load"/>
        <node name= "controller_spawner" pkg= "controller_manager" type="spawner" respawn="false"
output="screen" args="joint_state_controller robot_arm_controller hand_ee_controller"/>

        <!-- Robot State Publisher for TF of each joint: publishes all the current states of the joint, then RViz
can visualize -->
        <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
respawn="false" output="screen"/>

</launch>
```

Note: *change below parameters in the launch file code:*

- *Your_package_name: Name of your package. For my case, it will be "robot_arm_urdf"*
- *Your_Robot_name_defined_in_urdf_file: Name of your urdf file. I my case it is "robot_arm_urdf.urdf"*
- *Add the intial positions for all the joints available in your urdf. For each joint add "-J Joint_Name Position"*
- *In the "arg" attribute of "controller_spawner" node, give names of controllers you defined in the .yaml file. Add all the controller names saperated by space.*
- *If You used name space in controller: While writing the joint_trajectory_controller.yaml file, if you used a namespace, (please watch the you tube video to know more) you need to add that namespace in some tags here. Please find link to the video at the start of this section.*

6. Save and close the file.

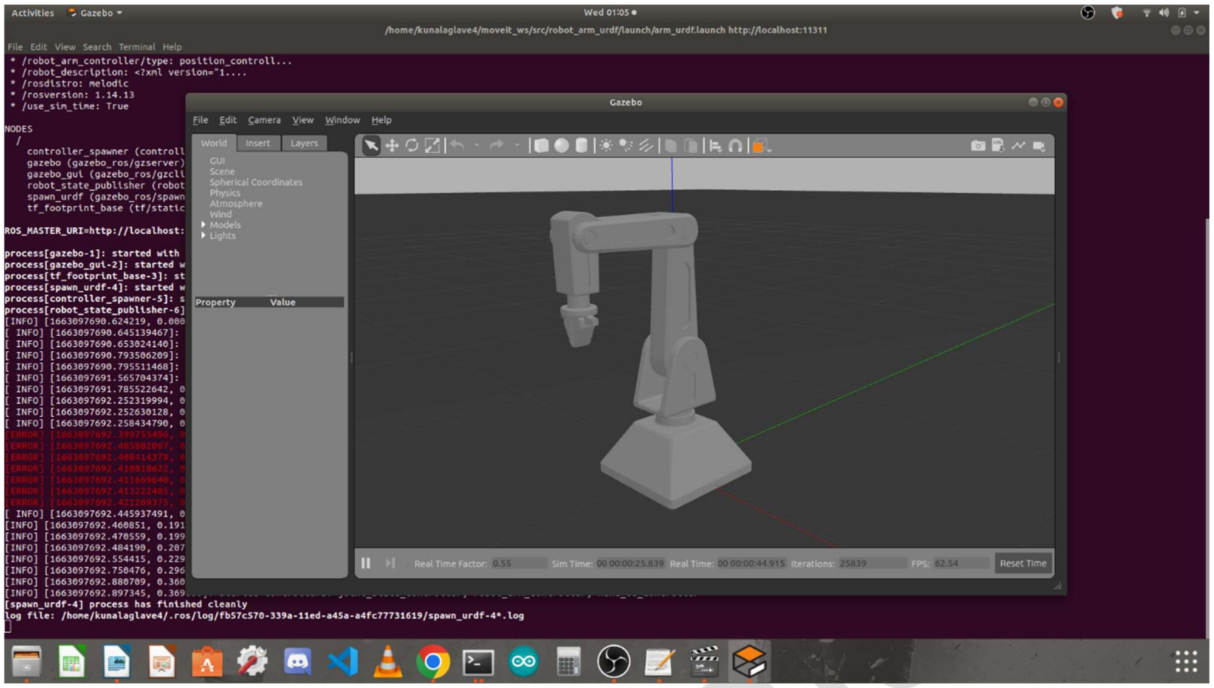
3.10 Build your catkin workspace

1. Open a terminal.
2. Go to your workspace
   ```
   $ cd ~/moveit_ws
   ```
3. Source the setup.bash file of your catkin work space.
   ```
   $ source devel/setup.bash
   ```
4. Build the workspace
   ```
   $ catkin build
   ```
5. Again, source the setup.bash file.
   ```
   $ source devel/setup.bash
   ```

3.11 Launch your robot's launch file to load it 1st time in GAZEBO.

1. Open a terminal and execute the following command to start the ROS master.
   ```
   $ roscore
   ```
   After running this command, keep this terminal open.
2. Open an another terminal.
3. Go to your workspace
   ```
   $ cd ~/moveit_ws
   ```
4. Source the setup.bash file of your catkin work space.
   ```
   $ source devel/setup.bash
   ```
5. Build the workspace if not built already after making the changes.
   ```
   $ catkin build
   ```
6. Again, source the setup.bash file.
   ```
   $ source devel/setup.bash
   ```
7. Launch the "arm_urdf.launch" file that we created in previous steps.
   ```
   $ roslaunch your_package_name launch_file_name.launch
   ```
   In my case, it will be:
   ```
   $ roslaunch robot_arm_urdf arm_urdf.launch
   ```

If you did everything right, your robot will open in the gazebo without any errors. Only the error related to PID gains may come. No other error should arrive. Also check if your all the controllers are spawned correctly.


Activities
Gazebo
Wed 01:05
/home/kunalaglave4/moveit_ws/src/robot_arm_urdf/launch/arm_urdf.launch http://localhost:11311
File Edit View Search Terminal Help
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Reset Time

```
 * /robot_arm_controller/type: position_controll...
 * /robot_description: <?xml version="1....
 * /rosdistro: melodic
 * /rosversion: 1.14.13
 * /use_sim_time: True

NODES
  /
    controller_spawner (controll
    gazebo (gazebo_ros/gzserver)
    gazebo_gui (gazebo_ros/gzcli
    robot_state_publisher (robot
    spawn_urdf (gazebo_ros/spawn
    tf_footprint_base (tf/static

ROS_MASTER_URI=http://localhost:

process[gazebo-1]: started with
process[gazebo_gui-2]: started w
process[tf_footprint_base-3]: st
process[spawn_urdf-4]: started w
process[controller_spawner-5]: s
process[robot_state_publisher-6]
[INFO] [1663097690.624219, 0.000
[ INFO] [1663097690.645139467]:
[ INFO] [1663097690.653024140]:
[ INFO] [1663097690.793506209]:
[ INFO] [1663097690.795511468]:
[ INFO] [1663097691.565704374]:
[ INFO] [1663097691.785522642, 0
[ INFO] [1663097692.252319994, 0
[ INFO] [1663097692.252630128, 0
[ INFO] [1663097692.258434790, 0
[ERROR] [1663097692.399755496, 0
[ERROR] [1663097692.405882067, 0
[ERROR] [1663097692.408414379, 0
[ERROR] [1663097692.410018622, 0
[ERROR] [1663097692.411669646, 0
[ERROR] [1663097692.413222485, 0
[ERROR] [1663097692.421269375, 0
[ INFO] [1663097692.445937491, 0
[INFO] [1663097692.460851, 0.191
[INFO] [1663097692.470559, 0.199
[INFO] [1663097692.484190, 0.207
[INFO] [1663097692.554415, 0.229
[INFO] [1663097692.750476, 0.296
[INFO] [1663097692.880709, 0.360
[INFO] [1663097692.897345, 0.369
[spawn_urdf-4] process has finished cleanly
log file: /home/kunalaglave4/.ros/log/fb57c570-339a-11ed-a45a-a4fc77731619/spawn_urdf-4*.log
```

Real Time Factor: 0.55 Sim Time: 00 00:00:25.839 Real Time: 00 00:00:44.915 Iterations: 25839 FPS: 62.54

4 References:

1. http://wiki.ros.org/catkin/CMakeLists.txt
2. http://wiki.ros.org/catkin/package.xml
3. http://wiki.ros.org/ROS/Tutorials