Importing Your ROS Package Generated from SOLIDWORKS in ROS (ROBOT OPERATING SYSTEM) and Simulating it Using Moveit and Moveit Setup Assistant

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

This Tutorial is for those who want to simulate there own Robotic Arm in “Robot Operating System” from Scratch.

https://www.youtube.com/channel/UCNaMh2qgE3nU2euCg0EJ8NQ

Contents

Introduction

This tutorial series in created for those, who want to simulate their own robotic Arm in ROS using Movit, Rviz and Gazebo. But, as per my experience, there are no or insufficient tutorials/guides, which can help to getting started from scratch.

I had learnt ROS during my academics and thought it would be great to document my knowledge so that, it can be helpful for other who want to learn ROS.

In this tutorial I have explained everything step by step starting from "Preparing the model for exporting as URDF from SOLIDWORKS" to "Simulating the exported URDF it in ROS".

You need to follow each and every tutorial to successfully simulate your robot arm. Though, I have not explained the technical details in deep, I have added some useful links in the PDF document for your reference. You can use those links to learn about the topics.

It was not possible to make a single video, as it is a very long but simple process. Once you start doing it, you will start understanding automatically.

All the video lessons are uploaded on YouTube here:
https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12

GitHub Repository to get the ready to use packages which I prepared in this tutorial:
https://github.com/kunalaglave4/import_your_custom_urdf_package_to_ROS.git

You can directly download the **“robot_arm_urdf”** and **“movit_robot_arm_sim”** package folder and place them in the SRC folder of your Catkin Workspace. You can follow the guidelines to which files to launch.

Currently, the code is not commented well. But, will update the well commented code soon.

Thank you for refereeing this tutorial. Hope it will be helpful to you and you will be able to simulate your own custom robot in ROS.

List of Video Lessons on YouTube

The full learning path for this tutorial series:

- **Lesson 1:** Assemble the Robotic Arm Properly in SOLIDWORKS: https://youtu.be/Bsh3DWmQ-uM
- **Lesson 2:** Set the zero positions of th joints and set axis systems: https://youtu.be/g7mZp8hnIok
- **Lesson 3:** Export the URDF file from SOLIDWORKS: https://youtu.be/I08lO_SRBbk
- **Lesson 4:** Send the Exported Package (URDF) to Linux and Create new Catkin Workspace: https://youtu.be/ZWliEJfNtlM
- **Lesson 5:** Add Your URDF Package to Catkin Workspace and add dependencies: https://youtu.be/m75_ZIitQPM
- **Lesson 6:** Modify the URDF file to make it suitable for Simulation: https://youtu.be/oRd7dPLYww0
- **Lesson 7:** Create a JointTrajectoryController to control your robotic arm's joints: https://youtu.be/3XHpYBB9WU0
- **Lesson 8:** Create a ".Launch" file to load your URDF and controllers in Gazebo: https://youtu.be/NY7f76m6xAM
- **Lesson 9:** Build the Catkin Workspace and launch the URDF for 1st time in Gazebo: https://youtu.be/QH-V2F0tMm8
- **Lesson 10:** Create a Miveit Package to manipulate(Adding Motion) your robotic arm using "Moveit Setup Assistant": https://youtu.be/DZB5_4JCS0A
- **Lesson 11:** Testing the Moveit Package by launching the default launch files: https://youtu.be/FM6zFxPXlmg
- **Lesson 12:** Replace the faulty ROS Controller YAML file with New ROS Controller file: https://youtu.be/bKziqIz2GCU
- **Lesson 13:** Create a ".launch" file to load full simulation: https://youtu.be/Fiaty-FX280
- **Lesson 14:** Launch the simulation and plan some motion: https://youtu.be/ThMhPcrZpgk

1 Conventions Used

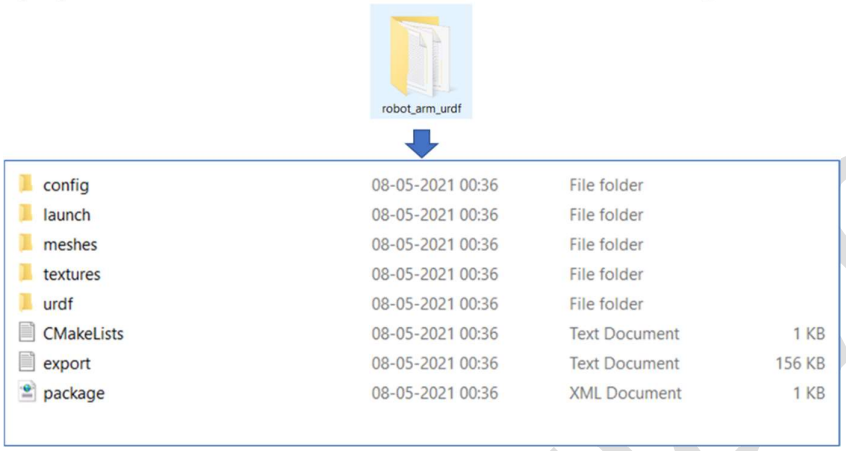
In this document, some colour coding is done to distinguish some notations. They are as given below

- **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.

  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.



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 motion of joints, their mass properties, inertial properties etc. Also, this folder contains a your_package_name.csv file containing information of your robot's 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 SOLIDWORKS directly to your Catkin Workspace. Once you copy it, build your Catkin Workspace and launch the gazeb.launch or display.launch file.

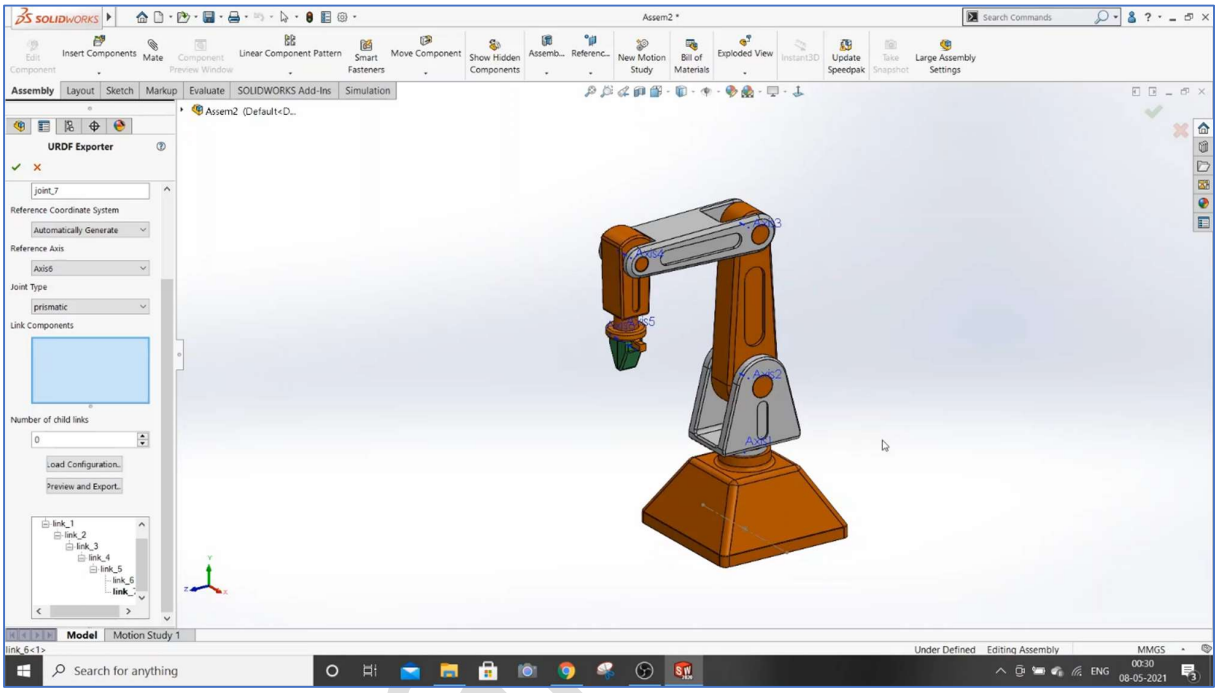
But the default generated package is just a robot description. If you don't modify it, you cannot give any motion commands, control the joints or any other thing.
In this tutorial, we will see, how you can modify the package generated from SOLIDWORKS and how to **simulate it using ROS, Rviz, Gazebo and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

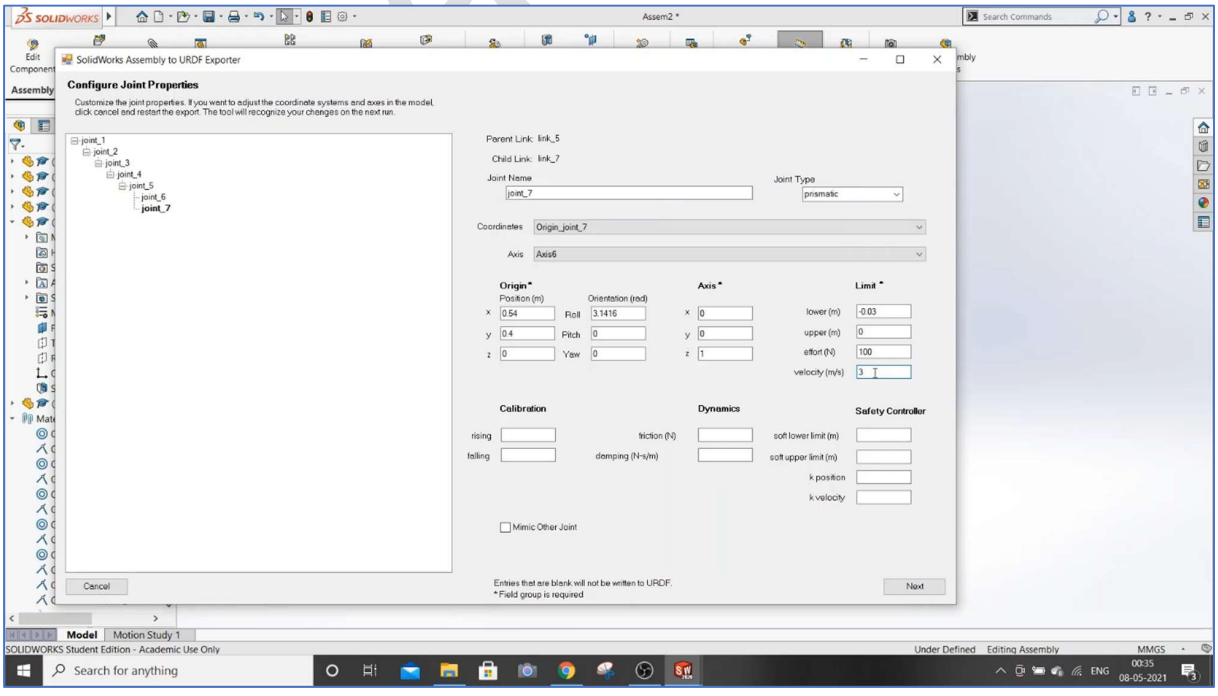
URDF File was Created in Previous Tutorials

Please follow the Lesson 1 to lesson 3 to know how to export URDF from SOLIDWORKS: https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12

In Lesson 1 to Lesson 3 we created the URDF using SOLIDWORKS from this Robotic Arm Model.

The Same URDF will be used in Lesson 4 onwards.





3 Importing the package/URDF generated from SOLIDWORKS in ROS

Link to the full YouTube video tutorial playlist:
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

Full YouTube Tutorial: https://youtu.be/ZWIiEJfNtIM

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

Full YouTube Tutorial Link: https://youtu.be/m75_ZIitQPM

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.
- One can use TAB for indentation.

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 Modify the URDF file to make it suitable for Simulation

Full YouTube Tutorial: https://youtu.be/oRd7dPLYww0

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 some 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 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 package's URDF folder and start making changes in it as given below:

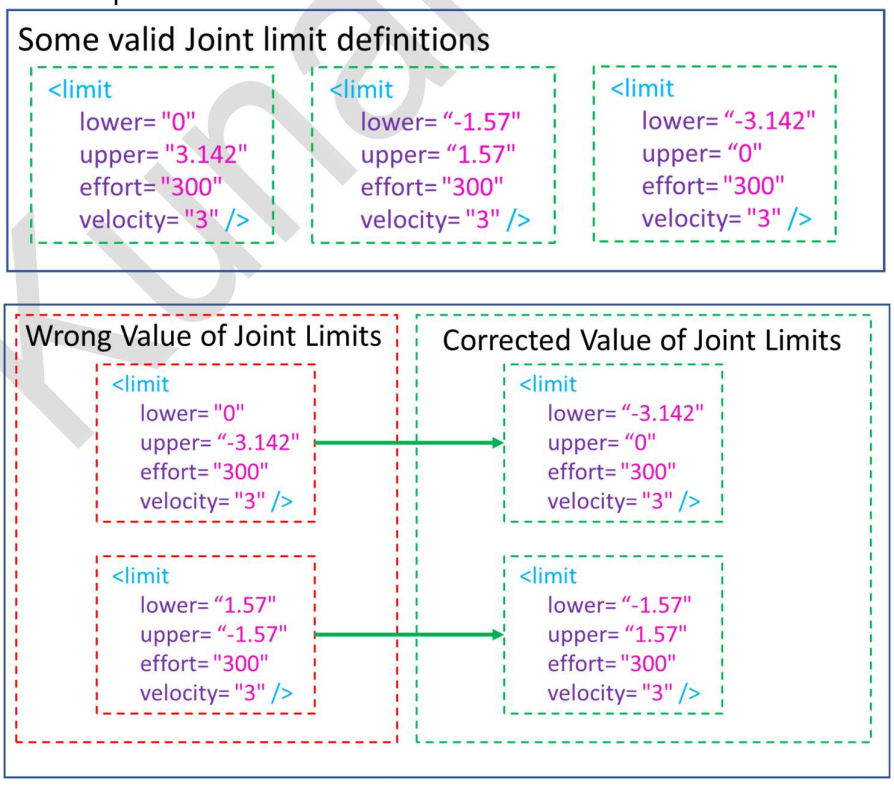
1. We need to add a "world" link and fix the "base_link" to it. Add below given code snippet in your URDF between the <robot name="robot_arm_urdf"> and the <link name="base_link"> tags as given below.

```
<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's 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:



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.

YouTube Tutorial Link: https://youtu.be/3XHpYBB9WU0

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 about controllers: 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 is.

```
#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 joint names of the arm (If any) and end effector (if any) 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 an indent level. One can use two spaces at the place of one TAB. (Watch the video for more details: https://youtu.be/3XHpYBB9WU0*

6. Save and close the file.

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

Full YouTube Video Tutorial: https://youtu.be/NY7f76m6xAM

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

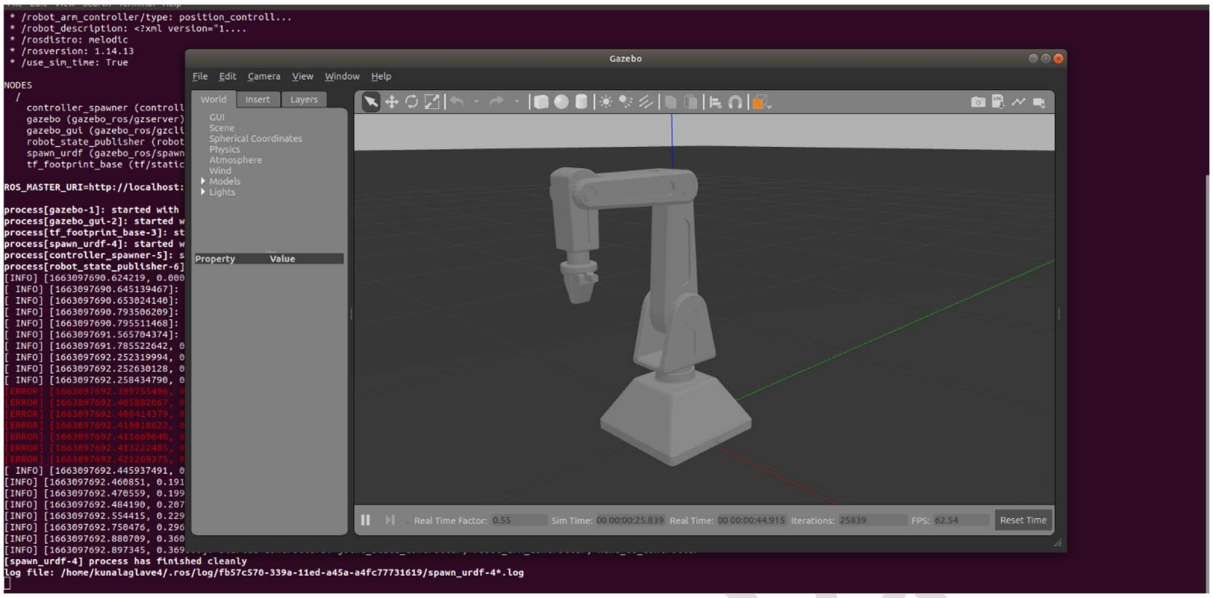
1. Open a terminal.
2. Go to your workspace
   $ cd ~/moveit_ws
3. Source the setup.bash file of your catkin work space.
   $ source devel/setup.bash
4. Build the workspace
   $ catkin build
5. Again, source the setup.bash file.
   $ source devel/setup.bash

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

Full YouTube Video Tutorial: https://youtu.be/QH-V2F0tMm8

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 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.


* /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
[INFO] [1663097692.44593749
[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] [1663097692.880789, 0.368
[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
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Real Time Factor: 0.55 Sim Time: 00 00:00:25.839 Real Time: 00 00:00:44.915 Iterations: 25839 FPS: 62.54
Reset Time

4 Creating a new Manipulator Package to control the Robotic Arm URDF using Moveit Setup Assistant

Moveit is a great open source Robot manipulation tool which can be used to create packages, which can be used to manipulate any Robot in ROS.

In this tutorial, the URDF file that we configured in chapter 3 of this document will be used.

Moveit official website: https://moveit.ros.org/

We will be using ROS 1 and Moveit 1 for this tutorial. Moveit is installed by default with the Full Desktop Version of any ROS distribution. Use below commands to install and configure the Moveit.

Command to Install Moveit in ROS melodic:

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

Install ROS Controllers:

```
$ sudo apt-get install ros- melodic -ros-control ros- melodic -ros-controllers
```

Note: for other distros, replace the name of distro with your ROS distribution name.

4.1 Creating the Moveit Package using Moveit Setup Assistant to control our URDF file.

Before starting this tutorial, make sure, you have configured your URDF file in proper way as given in chapter 3 of this document or Lesson 1 to Lesson 9 of this playlist:
https://youtube.com/playlist?list=PLeEzO_sX5H6TBD6EMGgV-qdhzxPY19m12

Full YouTube Tutorial: https://youtu.be/DZB5_4JCS0A

Important Note: I will not add steps for this tutorial here. Because, it is better to watch the video itself. The video also explains how to solve the errors in the URDF file like error in direction of motion of a robot joint etc. So, watch the full tutorial carefully.

Steps with commands used in the tutorial:

1. Launch a new terminal and go to your Catkin Workspace, where the URDF package is saved. (My Catkin workspace is created in home directory with name “moveit_ws”)
   ```
   $ cd ~/moveit_ws
   ```
2. Build the workspace if not already
   Those who use build command:
   ```
   $ catkin build
   ```
   Those who use make command:
   ```
   $ catkin_make
   ```
3. Source the new setub.bash file in the devel folder of the workspace.
   ```
   $ source devel/setup.bash
   ```
   Or
   ```
   $ source ~/moveit_ws/devel/setup.bash
   ```
4. Launch Moveit Setup Assistant
   ```
   $ roslaunch moveit_setup_assistant setup_assistant.launch
   ```

   Warning: Do not launch Moveit setup assistance before sourcing the setup.bash file of your workspace.
5. Now follow the steps as given in the video.

6. While generating the package from moveit, I named it as “movit_robot_arm_sim” (I types “movit” instead of “moveit” to make autocomplete easy. Because there are lot of packages starting with “moveit” keyword.) If you used different name, then use that package name in commands of further section.

4.2 Testing the Moveit Package Generated using Moveit Setup Assistance.

Full YouTube Tutorial: https://youtu.be/FM6zFxPXlmg

In previous step 4.1, we generated a manipulation package for our custom URDF which was generated from SOLIDWORKS.

We will check it first if it is generated properly or not. Because, the default ros_controller.yaml file generated by Moveit Setup Assistant is incomplete or can’t be used as it is to control our robot.

The Moveit Setup assistant has exported a package with all the data needed to control the robot arm. It has created some launch files for the 1st demo.

- **demo.launch:** It opens the robot in Rviz only.
- **gazebo.launch**: It opens the robot in gazebo only.
- **demo_gazebo.launch:** It opens the robot in Rviz as well as Gazebo. But, the Rviz and Gazebo simulations are not connected to each other due to incomplete Moveit controller or ROS controller.

Follow below steps:

1. First of all, we need to build our catkin workspace as the new package is added to it.
    a. Launch a new terminal.
    b. Go to your workspace
        $ cd ~/moveit_ws
    c. Source the setup.bash file
        $ source devel/setup.bash
    d. Build your workspace
        $ catkin build
        Or
        $ catkin_make
    e. Source the new setup.bash file again
        $ source devel/setup.bash
2. Now, launch the demo_gazebo.launch file
    $ roslaunch YourMoveitPackagename demo_gazebo.launch
    If you named the package same as my package (movit_robot_arm_sim), the command will be:
    $ roslaunch movit_robot_arm_sim demo_gazebo.launch
3. Wait for some time till everything launches. Wait till both Gazebo and Rviz windows appear.
4. Go to the terminal. You can see some errors along with the pid gain errors.
5. You can try to execute the position commands from Rviz as given in the video (Link given at the beginning of chapter 4.2). But it will give error in the terminal for controllers. This is because, the “ros_controller” for moveit simulation n “moveit_robot_arm_sim” package is not connected with the joint_trajectory_controller in the “robot_arm_urdf” package.
6. Go in the terminal. Click somewhere in the terminal and press “Ctrl + c” to terminate everything running.

In next chapter we will configure the package to solve the issues.

5 Configuring the Default Moveit Package to work properly.

The package generated using Moveit contains some incomplete information of the controllers. We need to configure it before starting to use.

5.1 Writing a new ROS Controller for connecting the joint_trajectory_controller with ROS manipulator or moveit package.

Full YouTube Tutorial: https://youtu.be/bKziqIz2GCU

1. Open a new terminal.
2. Go to the config folder of your moveit package that we created in step 4.1.
   My moveit package name is “movit_robot_arm_sim” (watch the spelling carefully) and workspace name is “moveit_ws”. So, the command will be as given below:
   $ cd ~/moveit_ws/src/movit_robot_arm/config
   If your workspace or package name is different, then command will be:
   $ cd ~/YourWorkSpaceName/src/YourMoveitPackageName/config
3. Type below command to create a “.yaml” file with name “new_ros_controllers.yaml” and press enter.
   $ touch new_ros_controllers.yaml
4. A new file will be created.
5. Type below command to open it in gedit text editor. (One can use this command directly without using “touch”. It will create the file if not existing already and open in editor)
   $ gedit new_ros_controllers.yaml
6. Copy and pest below lines of code as it is.

```
#This is a movit contoller connecting follow_joint_trajectory controller with
JointTrajectoryController

controller_list:
  - name: robot_arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
    - joint_1
    - joint_2
    - joint_3
    - joint_4
    - joint_5
  - name: hand_ee_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
    - joint_6
    - joint_7
```

Follow the indentation properly. Use two spaces at the place of TAB as the TAB is not supported in YAML file. Follow the video tutorial for better understanding.

https://www.youtube.com/channel/UCNaMh2qgE3nU2euCg0EJ8NQ

7. Once done, save and close the file.

Note:

1. If you used different names for the controllers written in the joint_trajectory_controller.yaml in the robot_arm_urdf package, then use the same names here. My joint_trajectory_controller.yaml file looks as given below:

joint_trajectory_controller.yaml file

```
#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
```

I used same names in the “new_ros_controllers.yaml” file. Bacuse, here, we are not creating new controller, but connecting the “FollowJointTrajectory” node/controller with the existing “JointTrajectoryController”. In the “new_ros_controllers.yaml” file, the “name” is the name of your “JointTrajecotyController” name which is created in the “robot_arm_urdf” and the “joint” attribute contains list of joints to control.

2. If you have extra joints in your robot, add the extra joint in their respective group (arm or end effector)

5.2 Edit the “moveit_controller_manager.launch.xml” file available in the launch folder of your moveit package.

Full YouTube Tutorial: Done in above video tutorial only.

As we created a new ros controller, we need to pass this new controller name to the controller manager file which is available in “launch” folder of your moveit package generated using moveit setup assistant. The naming syntax of this file is “YourRobotNameInUrdfFile_moveit_controller_manager.launch.xml”. This file launches the default controller generated by setup assistant. We need to replace with newly created controller.

1. Using file manager, go to the “launch” folder of your moveit package. Mine is in my “moveit_ws” workspac’s src folder.
2. Find the file with name “YourRobotNameInUrdfFile_moveit_controller_manager.launch.xml”. In my case, it is “robot_arm_urdf_moveit_controller_manager.launch.xml”
3. Change the name “ros_controllers.yaml” with “new_ros_controller.yaml” as given below.

robot_arm_urdf_moveit_controller_manager.launch.xml [Before Editing]

```
<launch>
  <!-- Define the controller manager plugin to use for trajectory execution -->
  <param name="moveit_controller_manager"
value="moveit_simple_controller_manager/MoveItSimpleControllerManager" />

  <!-- loads controller list to the param server -->
  <rosparam file="$(find movit_robot_arm_sim)/config/ros_controllers.yaml"/>
</launch>
```

robot_arm_urdf_moveit_controller_manager.launch.xml [After Editing]

```
<launch>
  <!-- Define the controller manager plugin to use for trajectory execution -->
  <param name="moveit_controller_manager"
value="moveit_simple_controller_manager/MoveItSimpleControllerManager" />

  <!-- loads controller list to the param server -->
  <rosparam file="$(find movit_robot_arm_sim)/config/new_ros_controllers.yaml"/>
</launch>
```

4. Save and close the file.

5.3 Create a new launch file to launch moveit simulation in Rviz and Gazebo.

Full YouTube Tutorial: https://youtu.be/Fiaty-FX280

Now, we need to create a new launch file to load:

1. The launch file robot_arm.launch created in the “robot_arm_urdf” package in chapter 3.9 of this document.
2. Launch the Moveit Group node “move_group.launch” created in moveit package by moveit setup assistant.
3. Load the moveit.rviz Rviz config file.

Follow below steps to create a launch file to load moveit group node, Raviz and gazebo simulation of our robot arm:

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

full_robot_arm_sim.launch

```
<launch>

  <!-- Launch Your robot arms launch file which loads the robot in Gazebo and spawns the
controllers -->
  <include file = "$(find robot_arm_urdf)/launch/YourUrdfLauchFile.launch" />

  <!-- Launch Moveit Move Group Node -->
  <include file = "$(find YourMoveitPackageName)/launch/move_group.launch" />

  <!-- Run Rviz and load the default configuration to see the state of the move_group node -->
  <arg name="use_rviz" default="true" />

  <include file="$(find YourMoveitPackageName)/launch/moveit_rviz.launch" if="$(arg
use_rviz)">
    <arg name="rviz_config" value="$(find YourMoveitPackageName)/launch/moveit.rviz"/>
  </include>


</launch>
```

If you have everything same as me, then directly use below code

full_robot_arm_sim.launch

```
<launch>

  <!-- Launch Your robot arms launch file which loads the robot in Gazebo and spawns the
controllers -->
  <include file = "$(find robot_arm_urdf)/launch/arm_urdf.launch" />

  <!-- Launch Moveit Move Group Node -->
  <include file = "$(find movit_robot_arm_sim)/launch/move_group.launch" />

  <!-- Run Rviz and load the default configuration to see the state of the move_group node -->
  <arg name="use_rviz" default="true" />

  <include file="$(find movit_robot_arm_sim)/launch/moveit_rviz.launch" if="$(arg use_rviz)">
    <arg name="rviz_config" value="$(find movit_robot_arm_sim)/launch/moveit.rviz"/>
  </include>

</launch>
```

6. Save and close the file.

5.4 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

5.5 Launch your robot's moveit launch file to load it moveit simulation with Rviz and Gazebo

Full YouTube Tutorial: https://youtu.be/ThMhPcrZpgk

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 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_Moveit_package_name moveit_launch_file.launch
   In my case, it will be:
   $ roslaunch movit_robot_arm_sim full_robot_arm_sim.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 you're all the controllers are spawned correctly.

Start planning the motion of robot as given in the video: https://youtu.be/ThMhPcrZpgk

Terminating the Simulation:

Go to the terminal which is running the simulation and press "Ctrl + c".

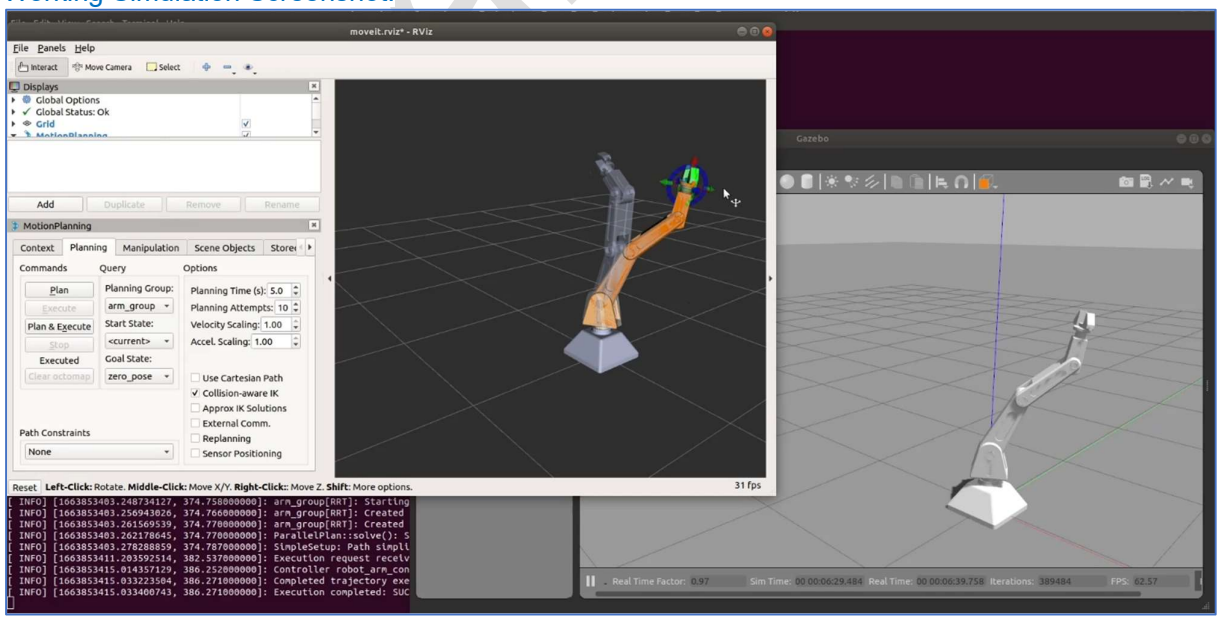
5.6 Solving the possible issues in after launching the simulation:

- If it gives any errors regarding controller after you execute any position:
  - Check the “new_ros_controllers.yaml” for spelling or indentation errors.
  - Also, check, if you added all the joints of the robot with their exact name in their respective group
  - Check if you added the name of new controller YAML file in the “moveit_controller_manager.launch.xml file.
- If you get error related to controller Manager:
  This happens because we built the moveit package 1st time with incomplete controller. When we build the workspace after modifying the controller, some caches may be remaining in the build files and causing the issue. You can solve it using below steps by freshly building the workspace.
  - Open a terminal.
  - Go to your workspace
    $ cd ~/moveit_ws
  - Source the setup.bash file of your catkin work space.
    $ source devel/setup.bash
  - Clean your workspace (Optional. Read the note below before running)
    $ catkin clean
  - Build the workspace
    $ catkin build
  - Again, source the setup.bash file.
    $ source devel/setup.bash
  - Try launching the simulation now.

Note:

*The **“catkin clean”** command deletes the build, devel and logs folders of the workspace. This ensures that the workspace builds freshly. If you have some other projects/packages which contains some important files in the build or devel folder, either backup them or do not run this command.*

Working Simulation Screenshot:



6 Further Steps

Using these packages, it is possible to do lot of things.

- The Robotic Arm can be controlled using Python Scripts called “Nodes”.
- Action servers can be created to give “Goals” to the robotic arm to complete specific tasks.
- This robotic arm can be added in any other ROS simulation to pick and place or any task.
- Real hardware, if available, can be controlled by connecting it with the simulation.
- And lot more can be done.

Stay connected for future updates.

Thank You!

7 References:

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