MoveIt! Tutorial for Custom Robot

In this tutorial, we will discuss how to setup a custom robot to be used with MoveIt! In ROS. This tutorial assumes you already have a working URDF file (we will be using a .urdf.xacro file in this example) that is setup to work properly with Gazebo (i.e joints, transmissions, and mass properties). Also, this is meant to be as generic as possible to be able to use with any custom robot, but you should know that every robot will be different, thus customization of the files and code mentioned in this tutorial will be necessary.

All files for this project can be obtained through the following repository:

https://github.com/mahumada/Moveit_tutorial

MoveIt! Setup Assistant

The setup assistant for MoveIt! will help make the files needed for MoveIt! to recognize and configure your robot. A detailed walkthrough of the setup assistant can be found on the MoveIt! tutorial section here. I won't reiterate what is said on the MoveIt! Tutorial, but all you should need is the .xacro file for your 'bot.

Rviz Setup

Next is to make sure that Rviz is working correctly with MoveIt! Luckily, there is another tutorial that tackles just that here.

Congrats! At this point you should have MoveIt! Controlling the 'bot in Rviz. The next steps will focus on integrating MoveIt! With your Dynamixel motors.

Dynamixel Setup and Meta Controller

Note: make sure you have the ROS stack dynamixel_motors installed, this will be the hardware interface between ROS and the actual Dynamixel actuators.

Please note that the information provided in this section was collected from both Mastering ROS (source code for Mastering ROS here) and the dynamixel_controllers tutorial. Feel free to browse both sources for more info. At this point, we will create a meta controller, that is to say one controller package that will be used to control multiple joints. First, you want to create a controller package that has a couple of dependencies. The following command will create the *awesomebot1000_controller* package with appropriate dependencies.

```
$ catkin_create_pkg awesomebot1000_controller roscpp rospy dynamixel_controllers std_msgs
sensor_msgs
```

The next step is to create a configuration file that will contain the parameters necessary for the dynamixel controllers. The following is the *awesomebot1000.yaml* file located in the *awesomebot1000_controller/config* folder:

```
joint1_controller:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: joint1
    joint_speed: 4
    motor:
        id: 6
        init: 512
        min: 0
        max: 1023

joint2_controller:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: joint2
    joint_speed: 4
    motor:
        id: 4
        init: 512
        min: 0
        max: 1023
```

Please note that this robot only has two joints, thus we only specify two joint controllers. Also, make sure that the motor id is set properly to the correct joint.

We need to create a configuration file that group up all controllers and make it an action server. Paste the text below into *awesomebot1000_trajectory_controller.yaml* in *awesomebot1000_controller/config*:

```
awesomebot1000_trajectory_controller:
   controller:
       package: dynamixel_controllers
       module: joint_trajectory_action_controller
       type: JointTrajectoryActionController
   joint_trajectory_action_node:
       min_velocity: 0.1
       constraints:
           goal_time: 0.25
```

After creating the JointTrajectory controller, we need to create a *joint_state_aggregator* node for combining and publishing the joint states of the robotic arm. You can find this node from the *awesomebot1000_controller/src* folder named *awesomebot1000_state_aggregator.cpp*. The function of this node is to subscribe controller states of each controller having message type of *dynamixel::JointState* and combine each message of the controller into the *sensor_msgs::JointState* messages and publish in the */joint_states* topic. This message will be the aggregate of the joint states of all the dynamixel controllers. Please copy the code from the repository listed above, no edits necessary.

After the .cpp file, you have to make sure to make it executable for ROS. In your awesomebot1000_controller package, open the *CMakeLists.txt* file. Locate the line that says:

```
## Declare a C++ executable
# add_executable(awesomebot1000_controller_node src/awesomebot1000_controller_node.cpp)
```

And uncomment the “add_executable” line, as well as add your joint_state_aggregator.cpp file so that it looks like the lines below:

```
## Declare a C++ executable
add_executable(awesomebot1000_state_aggregator src/awesomebot1000_state_aggregator.cpp)
```

Then, locate the line that says:

```
## Specify libraries to link a library or executable target against
# target_link_libraries(awesomebot1000_controller_node
#   ${catkin_LIBRARIES}
# )
```

And uncomment the “target_link” line and add your joint state aggregator dependency such that it looks like the lines below:

```
## Specify libraries to link a library or executable target against
target_link_libraries(awesomebot1000_state_aggregator ${catkin_LIBRARIES})
```

Next, create a launch file that will run the joint_state_aggregator node. Save the following code in *awesomebot1000_state_aggregator.launch* in the *awesomebot1000_controller/launch* folder, make sure to change the “pkg” name to the package you created, also list all the controllers you listed in the .yaml file above:

```
<?xml version="1.0"?>

<launch>
    <node name="awesomebot1000_state_aggregator" pkg="awesomebot1000_controller"
type="awesomebot1000_state_aggregator" output="screen">
    <rosparam>
            rate: 50
            controllers:
```

```
                    - joint1_controller
                    - joint2_controller
    </rosparam>
    </node>
</launch>
```

Next, we need to create a launch file that will start communication between the PC and the Dynamixel servos and start the controller manager. The controller manager parameters are serial port, baud rate, servo ID range, and update rate. Please paste the following code into *start_awesomebot1000_meta_controller.launch* (make sure to change the names to your 'bot):

```
<?xml version="1.0"?>

<launch>

    <!-- Start the Dynamixel motor manager to control all awesomebot1000 servos -->
   <node name="dynamixel_manager" pkg="dynamixel_controllers" type="controller_manager.py"
required="true" output="screen">
        <rosparam>
            namespace: dxl_manager
            serial_ports:
                dynamixel_port:
                    port_name: "/dev/ttyUSB0"
                    baud_rate: 1000000
                    min_motor_id: 0
                    max_motor_id: 6
                    update_rate: 20
        </rosparam>
    </node>

  <!-- Load joint controller configuration from YAML file to parameter server -->
  <rosparam file="$(find awesomebot1000_controller)/config/awesomebot1000.yaml"
command="load"/>

  <!-- Start all arm joint controllers -->
  <node name="controller_spawner" pkg="dynamixel_controllers" type="controller_spawner.py"
          args="--manager=dxl_manager
                --port dynamixel_port
                joint1_controller
                joint2_controller"
                  output="screen"/>

  <!-- Start the awesomebot1000 arm trajectory controller -->
       <rosparam file="$(find
awesomebot1000_controller)/config/awesomebot1000_trajectory_controller.yaml" command="load"/>
  <node name="controller_spawner_meta" pkg="dynamixel_controllers"
type="controller_spawner.py"
    args="--manager=dxl_manager
          --type=meta
          awesomebot1000_trajectory_controller
```

```
          joint1_controller
          joint2_controller"
          output="screen"/>

  <!-- Publish combined joint info -->
  <include file="$(find awesomebot1000_controller)/launch/joint_state_aggregator.launch" />

    <param name="robot_description" command="$(find xacro)/xacro.py '$(find
awesomebot1000)/urdf/awesomebot.urdf.xacro'" />
    <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" output="screen">
      <rosparam param="source_list">[joint_states]</rosparam>
      <rosparam param="use_gui">FALSE</rosparam>
    </node>

  </launch>
```

We need to make a *controllers.yaml* file in *awesomebot1000_moveit/config* so that MoveIt! Knows to use the controllers we specified. Paste the following code in *controllers.yaml*:

```
controller_list:
  - name: awesomebot1000_trajectory_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - joint1
      - joint2
```

Almost done, insert the following code into *awesomebot1000_moveit_controller_manager.launch.xml*, which was automatically made by MoveIt, in the *awesomebot1000_moveit/launch* folder:

```
<launch>
<!--
 Set the param that trajectory_execution_manager needs to find the controller plugin
-->
<arg name="moveit_controller_manager"
default="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>

<param name="moveit_controller_manager" value="$(arg moveit_controller_manager)"/>

<!--  load controller_list  -->

<rosparam file="$(find awesomebot1000_moveit)/config/controllers.yaml"/>
</launch>
```

Final step!! Open the demo.launch file in the MoveIt directory that was automatically created by MoveIt! *awesomebot1000_moveit/launch.* You need to change the line that reads:

```
    <arg name="fake_execution" value="true"/>
```

To:

```
    <arg name="fake_execution" value="false"/>
```

This will tell MoveIt! to execute the path on the hardware, not just in visualization.

All you need to do now is launch two launch files. First, launch the meta controller with the following:

```
roslaunch awesomebot1000_controller start_awesomebot1000_meta_controller.launch
```

Then, launch MoveIt! With the following (note: I renamed my demo.launch to awesome.launch):

```
roslaunch awesomebot1000_moveit awesome.launch
```