MoveIt tutorial

MoveIt is a useful plugin for calculating DK and IK for varying systems. As we have 5-DOF planar robot, it takes a lot of calculations to move a robot to some desired location. Thanks to MoveIt it becomes much easier.

To install MoveIt:

```
sudo apt install ros-kinetic-moveit
```

Or melodic instead of kinetic (depending on the version of ROS)

There is already a package that deals with MoveIt called moveit_arm; but if you want (recommended) to configure your own package then follow the steps below.

First, create a new moveit_robot.urdf.xacro file in robot_description folder.

Copy and paste the text from robot_description/urdf/robot.urdf.xacro

Basically, this is the same .urdf file, but at the end change from **hardware_interface/EffortJointInterface** to **hardware_interface/PositionJointInterface** in every transmission**.** We needed effort controllers to practice with PID gains, but now we can use position controllers.

Then, we have to configurate the package for specifically our robot:

1. Go to https://ros-planning.github.io/moveit_tutorials/doc/setup_assistant/setup_assistant_tutorial.html and follow the steps (go to step 1 directly, source your workspace first, you will run roslaunch moveit_setup_assistant setup_assistant.launch)
2. Create New Moveit Configuration Package and then navigate to robot_description package and moveit_robot.urdf.xacro file
3. Follow the steps from the tutorial: no virtual joints; you can add either one planning group for all joints or two including arm (m2m, joint2,4,6 except the last one) and end-effector (the last joint - “end”); choose KDL plugin; robot pose as you want; skip the step end-effectors, no passive joints; we already have gazebo simulation so skip this step; ROS control we will configure ourself, skip this step; and finally, choose path to your new package. Exit the assistant.

4. Check if everything configured correctly by launching rviz: roslaunch your_package demo.launch

5. https://ros-planning.github.io/moveit_tutorials/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html to follow Rviz visualization; if everything is ok, then MoveIt will calculate the path and simulate the movement

6. Now, we need to connect our gazebo controllers with MoveIt

   6.1 First, we need to establish new controllers called *JointTrajectoryControllers* for our planning groups; as I established two groups, I need to create two trajectory controllers. You can see them in ***robot_control/config/moveit_robot_control.yaml***. Do the same for your setup. The names for the groups must be the same as you named in the steps above (copy the file robot_control/moveit_robot_control.yaml and name it as you want).

   6.2 Now, create a new .launch file (copy and paste to the same folder Ayan's file moveit_robot_control.launch) in *robot_control* package similar to ***robot_control/launch/moveit_robot_control.launch.***

   The crucial part is: **<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" ns="/robot" output="screen" args="joint_state_controller NAME_OF_YOUR_GROUP"/>**

<rosparam file="$(find robot_control)/config/YOU_YAML_FILE.yaml" command="load" />

   6.3 Now, we need to modify or create a new .launch file (again just copy paste and modify) in gazebo_robot that refers to move_robot_control.launch instead of old control file
   Basically change from `<include file="$(find robot_control)/launch/robot_control.launch" />` to `<include file="$(find robot_control)/launch/YOUR_LAUNCH_FILE.launch" />`

```
<param name="robot_description" command="$(find xacro)/xacro.py '$(find robot_description)/YOUR_CREATED_URDF_FILE.urdf.xacro'"/>
```

   delete the

```
 <param name="robot_description1" command="$(find xacro)/xacro.py '$(find robot_description)/urdf/box.xacro'"/>
<node name="urdf_spawner1" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen" args="-urdf -model robot1 -param robot_description1" />
```

7. After completing configuring gazebo simulation, we need to modify some files in moveit package

   7.1 First, add new file called joint_names.yaml into ***your_moveit_package/config***

with the following content:
controller_joint_names: [motortom2m, joint2, joint4, joint6, end]

7.2 Also, in the same folder create new file called ***controllers.yaml*** with the same content as in ***moveit_arm/config/controllers.yaml,*** only change the names of the controllers group

7.3 Change content of
hand_tutorial_moveit_controller_manager.launch.xml (open with gedit) in launch folder to
<launch>
<rosparam file="$(find
YOUR_PACKAGE_NAME)/config/controllers.yaml"/>
<param name="use_controller_manager" value="false"/>
<param name="trajectory_execution/execution_duration_monitoring"
value="false"/>
<param name="moveit_controller_manager"
value="moveit_simple_controller_manager/MoveItSimpleControllerManage
r"/>
</launch>

7.4 Create a new file called ***moveit_planning_execution.launch*** in
YOUR_PACKAGE folder
and add:
<launch>
	# This is needed for gazebo execution
	<rosparam command="load" file="$(find
your_moveit_package)/config/joint_names.yaml"/>

	<include file="$(find
YOURMOVEITPACKAGE)/launch/planning_context.launch">
		<arg name="load_robot_description" value="true"/>
	</include>

 # The planning and execution components of MoveIt! configured to
 # publish the current configuration of the robot (simulated or real)
 # and the current state of the world as seen by the planner
 <include file="$(find
YOURMOVEITPACKAGE)/launch/move_group.launch">
   <arg name="publish_monitored_planning_scene" value="true" />
 </include>

```
        # This is needed for gazebo execution
        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
        <param name="/use_gui" value="false"/>
        <rosparam param="/source_list">[robot/joint_states]</rosparam>
        </node>

  # The visualization component of MoveIt!
  <include file="$(find
YOURMOVEITPACKAGE)/launch/moveit_rviz.launch">
        <arg name="config" value="true"/>
  </include>

</launch>
```

7.5 Open file yourmoveitpackage/moveit_rviz.launch and verify that you have this

```
  <arg name="config" default="false" />
  <arg unless="$(arg config)" name="command_args" value="" />
  <arg     if="$(arg config)" name="command_args" value="-d $(find
YOURMOVEITPACKAGENAME)/launch/moveit.rviz" />

  <node name="$(anon rviz)" launch-prefix="$(arg launch_prefix)"
pkg="rviz" type="rviz" respawn="false"
        args="$(arg command_args)" output="screen">
    <rosparam command="load" file="$(find
YOURMOVEITPACKAGENAME)/config/kinematics.yaml"/>
  </node>
```

8. So, now, basically everything is ready. (compile your catkin workspace just to make sure that all is still working). Launch your newly created moveit_gazebo.launch file (roslaunch gazebo_robot YOURNAMEOFTHE GAZEBOLAUNCHFILE.launch). It should load your JointTrajectoryControllers. Open new terminal and type **rostopic list**, to check that controllers are running

9. Then, **roslaunch your_moveit_package moveit_planning_execution.launch** file. Rviz will show up, in rviz try to move the robot and then click plan and execute button. If everything is ok, then the robot will move both in Rviz and in Gazebo.

Moving the robot from script

We can move robot using Rviz, but now we will learn how to do it from script.
In this tutorial I used cpp language.
First create a new package by **catkin_create_pkg name roscpp rospy**
In the src folder, create a new file called test.cpp
Open this file in any IDE you prefer, and copy and past this code:

```
#include <moveit/move_group_interface/move_group.h>
#include <moveit/planning_scene_interface/planning_scene_interface.h>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit_msgs/DisplayRobotState.h>
#include <moveit_msgs/DisplayTrajectory.h>
#include <moveit_msgs/AttachedCollisionObject.h>
#include <moveit_msgs/CollisionObject.h>
#include <moveit_visual_tools/moveit_visual_tools.h>
// Main moveit libraries are included
int main(int argc, char **argv)
{
  ros::init(argc, argv, "move_group_interface_tutorial");
  ros::NodeHandle node_handle;
  ros::AsyncSpinner spinner(0);
  spinner.start(); // For moveit implementation we need AsyncSpinner, we
cant use ros::spinOnce()
  static const std::string PLANNING_GROUP = "group1"; /* Now we
specify with what group we want work,
  here group1 is the name of my group controller*/
  moveit::planning_interface::MoveGroupInterface
move_group(PLANNING_GROUP); // loading move_group

  const robot_state::JointModelGroup *joint_model_group =
      move_group.getCurrentState()-
>getJointModelGroup(PLANNING_GROUP); //For joint control
  geometry_msgs::PoseStamped current_pose;
  geometry_msgs::PoseStamped target_pose; // Pose in ROS is implemented
using geometry_msgs::PoseStamped, google what is the type of this msg
  current_pose = move_group.getCurrentPose(); /* Retrieving the
information about the
  current position and orientation of the end effector*/
  target_pose = current_pose;
```

```
  target_pose.pose.position.x = target_pose.pose.position.x - 0.1; /* Basically
our target pose is the same as current,
  except that we want to move it a little bit along x-axis*/
  ros::Rate loop_rate(50); //Frequency
  while (ros::ok()){
    move_group.setApproximateJointValueTarget(target_pose); // To
calculate the trajectory
    move_group.move(); // Move the robot
    current_pose = move_group.getCurrentPose();
    if (abs(current_pose.pose.position.x - target_pose.pose.position.x) < 0.01)
{
      break; // Basically, check if we reached the desired position
    }
    loop_rate.sleep();
  }

  ROS_INFO("Done");
  ros::shutdown();
  return 0;
}
```

So, what does this script do? Basically, it moves the robot a little bit backwards along x-axis. So we can control the robot in the Cartesian space thanks to moveit from nodes.
Now, we need to configure our CmakeLists.txt:
Open it, then uncomment #add_compile_options(-std=c++11) to allow to use c++11.
In find_package() add moveit_ros_planning_interface. For example:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  message_generation
  moveit_ros_planning_interface
)
```

In ## Specify additional locations of header files
Your package locations should be listed before other locations

```
include_directories(
# include
 #${catkin_INCLUDE_DIRS} Uncomment this line
)
```

And finally, add executables:
add_executable(test_moveit src/test.cpp)
target_link_libraries(test_moveit ${catkin_LIBRARIES})

Then, compile it. Before running we should launch robot in Gazebo and moveit in Rviz. Then, we can run our node. The robot should move in Gazebo.