Build a MoveIt! Package

> In this exercise, we will create a MoveIt! package for an industrial robot. This package creates the configuration and launch files required to use a robot with the MoveIt! Motion-Control nodes. In general, the MoveIt! package does not contain any C++ code.

Motivation

MoveIt! is a free-space motion planning framework for ROS. It's an incredibly useful and easy-to-use tool for planning motions between two points in space without colliding with anything. Under the hood MoveIt is quite complicated, but unlike most ROS libraries, it has a really nice GUI Wizard to get you going.

Reference Example

Using MoveIt with ROS-I

Further Information and Resources

MoveIt's Standard Wizard Guide

Scan-N-Plan Application: Problem Statement

In this exercise, you will generate a MoveIt package for the UR5 workcell you built in a previous step. This process will mostly involve running the MoveIt! Setup Assistant. At the end of the exercise you should have the following:

1. A new package called `myworkcell_moveit_config`
2. A moveit configuration with one group ("manipulator"), that consists of the kinematic chain between the UR5's `base_link` and `tool0`.

Scan-N-Plan Application: Guidance

1. Start the MoveIt! Setup Assistant (don’t forget auto-complete with tab):

```
roslaunch moveit_setup_assistant setup_assistant.launch
```

2. Select “Create New MoveIt Configuration Package”, select the `workcell.xacro` you created previously, then “Load File”.
3. Work your way through the tabs on the left from the top down.

   1. Generate a self-collision matrix.
   2. Add a fixed virtual base joint. e.g.

   ```
   name = 'FixedBase' (arbitrary)
   child = 'world' (should match the URDF root link)
   parent = 'world' (reference frame used for motion planning)
   type = 'fixed'
   ```

   3. Add a planning group called `manipulator` that names the kinematic chain between `base_link` and `tool0`. Note: Follow ROS naming guidelines/requirements and don’t use any whitespace, anywhere.

      a. Set the kinematics solver to `KDLKinematicsPlugin`
   4. Create a few named positions (e.g. “home”, “allZeros”, etc.) to test with motion-planning.
   5. Don’t worry about adding end effectors/grippers or passive joints for this exercise.
   6. Enter author / maintainer info.

      Yes, it’s required, but doesn’t have to be valid
   7. Generate a new package and name it `myworkcell_moveit_config`.

      - make sure to create the package inside your `catkin_ws/src` directory

The outcome of these steps will be a new package that contains a large number of launch and configuration files. At this point, it’s possible to do motion planning, but not to execute the plan on any robot. To try out your new configuration:

```
catkin build
source ~/catkin_ws/devel/setup.bash
roslaunch myworkcell_moveit_config demo.launch
```

Don’t worry about learning how to use RViz to move the robot; that’s what we’ll cover in the next session!

Using MoveIt! with Physical Hardware

MoveIt!'s setup assistant generates a suite of files that, upon launch:

- Loads your workspace description to the parameter server.
- Starts a node `move_group` that offers a suite of ROS services & actions for doing kinematics, motion planning, and more.
- An internal simulator that publishes the last planned path on a loop for other tools (like RViz) to visualize.

Essentially, MoveIt can publish a ROS message that defines a trajectory (joint positions over time), but it doesn't know how to pass that trajectory to your hardware.

To do this, we need to define a few extra files.

1. Create a `controllers.yaml` file (`myworkcell_moveit_config/config/controllers.yaml`) with the following contents:

```
controller_list:
  - name: ""
    action_ns: joint_trajectory_action
    type: FollowJointTrajectory
    joints: [shoulder_pan_joint, shoulder_lift_joint, elbow_joint, wrist_1_joint,
wrist_2_joint, wrist_3_joint]
```

2. Create the `joint_names.yaml` file (`myworkcell_moveit_config/config/joint_names.yaml`):

```
controller_joint_names: [shoulder_pan_joint, shoulder_lift_joint, elbow_joint,
wrist_1_joint, wrist_2_joint, wrist_3_joint]
```

3. Fill in the existing, but blank, controller_manager launch file (`myworkcell_moveit_config/launch/myworkcell_moveit_controller_manager.launch.xml`):

```
<launch>
  <arg name="moveit_controller_manager"
       default="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>
  <param name="moveit_controller_manager"
         value="$(arg moveit_controller_manager)"/>

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

4. Create a new `myworkcell_planning_execution.launch` (in `myworkcell_moveit_config/launch`):

```
<launch>
  <!-- The planning and execution components of MoveIt! configured to run -->
  <!-- using the ROS-Industrial interface. -->

  <!-- Non-standard joint names:
       - Create a file [robot_moveit_config]/config/joint_names.yaml
           controller_joint_names: [joint_1, joint_2, ... joint_N]
       - Update with joint names for your robot (in order expected by rbt controller)
       - and uncomment the following line: -->
  <rosparam command="load" file="$(find myworkcell_moveit_config)/config
/joint_names.yaml"/>

  <!-- the "sim" argument controls whether we connect to a Simulated or Real robot -->
  <!--  - if sim=false, a robot_ip argument is required -->
  <arg name="sim" default="true" />
  <arg name="robot_ip" unless="$(arg sim)" />

  <!-- load the robot_description parameter before launching ROS-I nodes -->
  <include file="$(find myworkcell_moveit_config)/launch/planning_context.launch" >
   <arg name="load_robot_description" value="true" />
  </include>

  <!-- run the robot simulator and action interface nodes -->
  <group if="$(arg sim)">
    <include file="$(find industrial_robot_simulator)/launch
/robot_interface_simulator.launch" />

    <!-- publish the robot state (tf transforms) -->
    <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
  </group>

  <!-- run the "real robot" interface nodes -->
  <!--   - this typically includes: robot_state, motion_interface, and
joint_trajectory_action nodes -->
  <!--   - replace these calls with appropriate robot-specific calls or launch files -->
  <group unless="$(arg sim)">
    <remap from="follow_joint_trajectory" to="joint_trajectory_action"/>
    <include file="$(find ur_modern_driver)/launch/ur_common.launch" >
      <arg name="robot_ip" value="$(arg robot_ip)"/>
      <arg name="min_payload" value="0.0"/>
      <arg name="max_payload" value="5.0"/>
    </include>
  </group>

  <include file="$(find myworkcell_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <include file="$(find myworkcell_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>

</launch>
```

5. Now let’s test the new launch files we created:

```
roslaunch myworkcell_moveit_config myworkcell_planning_execution.launch
```