ROS Exercise 3 - Path Planning and MoveIT

Mobile Robotics

Prof. Dr.-Ing. Gavin Kane

HS Emden Leer
Fachbereich Technik
Abteilung Elektrotechnik + Informatik

Overview In this practical you will:

- use MoveIT to create a Path Planner for a robot (Group),
- use the MoveIT RViz plugin,
- use C++ / Python to interface with the MoveIT Planning Groups.

1 MoveIT Start up Wizard

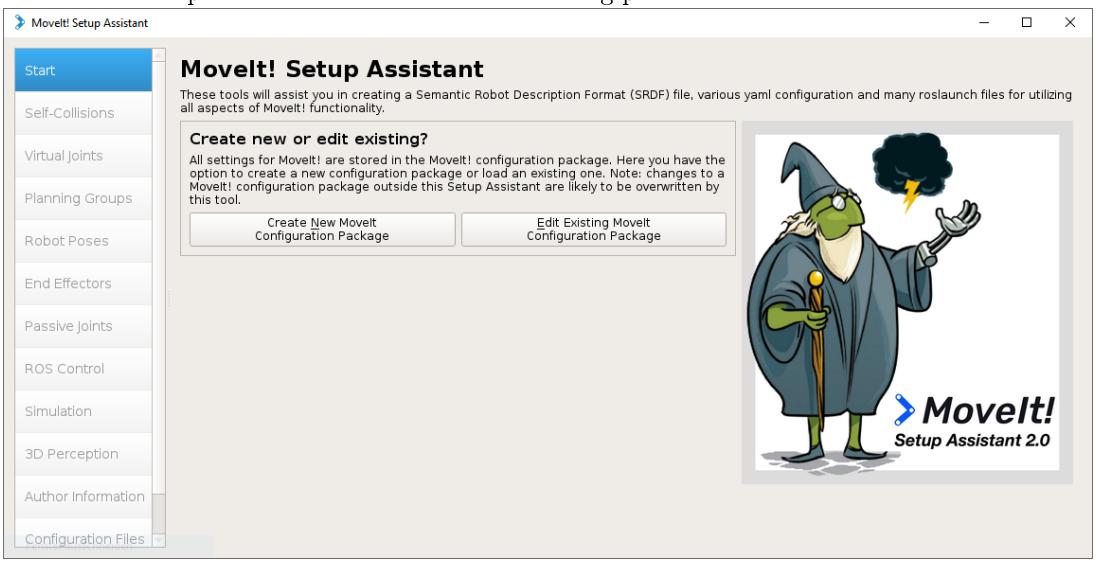
In order to work with the MoveIT System, it first needs to be installed. Run:

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

After the installation is complete, start up the Wizard with:

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

This should start up the wizard as seen in the following picture:



We will be building a Moveit Package using the robot scene that we created last week. Because no Moveit Package already exists for this robot, click on the "Create New Moveit Configuration Package" Button.

The next screen should give you an option to load in the URDF oder XACRO model. Click on the Browse button, and navigate to the scene that you created at the end of the tutorial last week. When this is selected, click on the "Load Files" button, in the bottom right.

When the URDF / XACRO Practical from last week was completed properly, the wizard should have no problems loading the scene. When this is completed, a 3D view of the scene should be visible on the

right hand side of the wizard.

We will now work through the different stages in setting up the planner. Working from the top to bottom, on the left hand side of the page, the following steps should be completed:

- **Self-Collisions**: The Wizard will determine which links can possibly collide with which links. Move the slider to a higher sampling density, and click Generate Collision Matrix. Here you will see the wizard has determined which collisions it can ignore in it's movement planning and which collisions need to be calculated.
- **Virtual Joints**: This is not necessary for our model, because our XACRO includes a base link "world". Should we have loaded a basic URDF for just a robot, we could include a virtual joint here, joining a robot to the world.
- **Planning Groups**: Here we need to assign the joints in the scene to the planning groups. By Clicking on "Add Groups", providing a name for your robot "robot1". For the Kinematics Solver, choose the *dl_kinematics_plugin/KDLKinematicsPlugin*. For OMPL Default Planner choose the "RRT". Then click on "Add Joints", and selecting all of the joints of one of the robots, move them to the planning group with the ">" button. Click Save. Repeat this for the other Robots, so that one planning group exists for each robot.
- **Robot Poses**: For this tutorial, add in three different Poses for each Robot:
  - "Candle", the robot is positioned with all joints straight up.
  - "HandOff", the robot is orientated towards another robot, as if it is going to pass something the the next robot.
  - "PickUp", the end-effector of the robot is lowered towards the ground, as if an end-effector could pick up an object.
- **End-effectors**: Select the final link of each robot.
- **Author Information**: Fill out as appropriate.
- **Configuration Files**: In order to create a new package folder inside the "src" directory, provide an appropriate name. Make sure you do not leave it as the "urdf_exercise" directory from where you loaded the XACRO file. Finally click, "Generate Package"

2 MoveIT RViz plugin

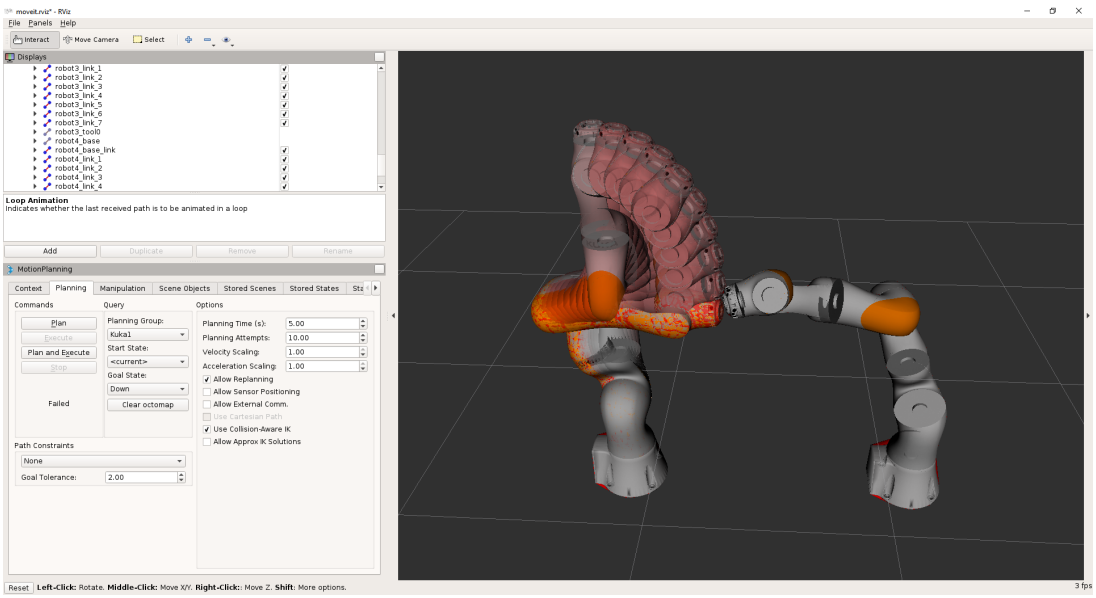
Compile the newly created package with catkin build and because a new package has been created, do not forget to source your workspace.

Then startup RVIZ with the new scene and Movement planner with the command:

```
$ roslaunch urdf_exercise_moveit demo.launch
```

where, "urdf_exercise_moveit is the package name that was created in the last step of the Moveit Wizard.

This should start up RVIZ, with the motion planner already running in the bottom left. Here it is possible to select a planning group, give the robot a new Goal State, and by clicking on " Plan and Execute" The robot should be seen moving in a colision free motion to the goal state.



It may be useful to disable the "Loop Animation" in "Planned Path" in the Tree Structure.

3 Moveit Commander

While the RVIZ demo is running, execute the following command in a further Terminal window:

```
$ rosrun moveit_commander moveit_commander_cmdline.py
```

This starts a commander that allows control of the robots through the command line. Changing Planning Groups with "use", go to pre-planned position with "go" and the pose name. You can also adjust the position with "go down 0.1" as an example to go down (negative z direction) 0.1m.

```
[ INFO] [1559079394.462633400]: Loading robot model 'myscene'...
[ INFO] [1559079394.463943100]: No root/virtual joint specified in SRDF. Assuming fixed joint

Waiting for commands. Type 'help' to get a list of known commands.

> use Kuka1
[ INFO] [1559079434.163605700]: Ready to take commands for planning group Kuka1.
OK
Kuka1> go Candle
Moved to Candle
Kuka1> go DownPos
Moved to DownPos
Kuka1> use Kuka2
[ INFO] [1559079471.561343400]: Ready to take commands for planning group Kuka2.
OK
Kuka2> go DownPos
Moved to DownPos
Kuka2> go Candle
Moved to Candle
```

4 MoveIT API - Python Version

In order to access the Moveit Commander Programmatically, it is necessary to start a node that accesses the moveit commander action server and requests planning trajectories.

The following Python code is an example using the previously planned poses and groups:

```
#!/usr/bin/env python

## To use the python interface to move_group, import the moveit_commander
## module.  We also import rospy and some messages that we will use.
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
```

```
import actionlib
import geometry_msgs

def simple_move():
  ## First initialize moveit_commander and rospy.
  moveit_commander.roscpp_initialize(sys.argv)
  rospy.init_node('simple_move',
                  anonymous=True)

  ## Instantiate a MoveGroupCommander object.  This object is an interface
  ## to one group of joints.  In this case the group refers to the joints of
  ## robot1. This interface can be used to plan and execute motions on robot1.
  robot1_group = moveit_commander.MoveGroupCommander("Kuka1")
  ## MoveGroup Commander Object for robot2.
  robot2_group = moveit_commander.MoveGroupCommander("Kuka2")

  ## Action clients to the ExecuteTrajectory action server.
  robot1_client = actionlib.SimpleActionClient('execute_trajectory',
    moveit_msgs.msg.ExecuteTrajectoryAction)
  robot1_client.wait_for_server()
  rospy.loginfo('Execute Trajectory server is available for robot1')
  robot2_client = actionlib.SimpleActionClient('execute_trajectory',
    moveit_msgs.msg.ExecuteTrajectoryAction)
  robot2_client.wait_for_server()
  rospy.loginfo('Execute Trajectory server is available for robot2')

  ## Set a named joint configuration as the goal to plan for a move group.
  ## Named joint configurations are the robot poses defined via MoveIt! Setup Assistant.
  robot2_group.set_named_target("Candle")

  ## Plan to the desired joint-space goal using the default planner (RRTConnect).
  plan = robot1_group.plan()
  ## Create a goal message object for the action server.
  robot2_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
  ## Update the trajectory in the goal message.
  robot2_goal.trajectory = plan

  ## Send the goal to the action server.
  robot2_client.send_goal(robot1_goal)
  robot2_client.wait_for_result()

  robot2_group.set_named_target("Down")

  plan = robot1_group.plan()
  robot2_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
  robot2_goal.trajectory = plan

  robot1_client.send_goal(robot1_goal)
  robot1_client.wait_for_result()


if __name__=='__main__':
  try:
    simple_move()
  except rospy.ROSInterruptException:
    pass
```

After saving this code in the scripts directory of the urdf_exercise package, it can be run with the following command:

```
rosrun urdf_exercise_moveit simple_move.py
```

5 Practical Goal

Write a Python Script, such that the robots hand off a virtual item between themselves. Waiting at the candle position when not required.