Collision Detection

Preface

We have built MoveIt environments on both the Jetson_Nano motherboard and the Orin series motherboard. Due to the onboard performance of the Jetson_Nano, running the MoveIt program on the motherboard will be more laggy and slow to load, and it will take about 3 minutes to complete the loading. Therefore, we recommend that users of the Jetson motherboard run the MoveIt program on the configured virtual machine we provide. The Orin motherboard can run MoveIt smoothly on the motherboard without running it on a virtual machine. Whether running on a virtual machine or on the motherboard, the startup instructions are the same. The following tutorials will take running on the Orin motherboard as an example.

1. Functional Description

After the program runs, rviz will add an obstacle next to the robotic arm, and the robotic arm will make random movements. During the motion planning process, the robotic arm will not collide with the added obstacle.

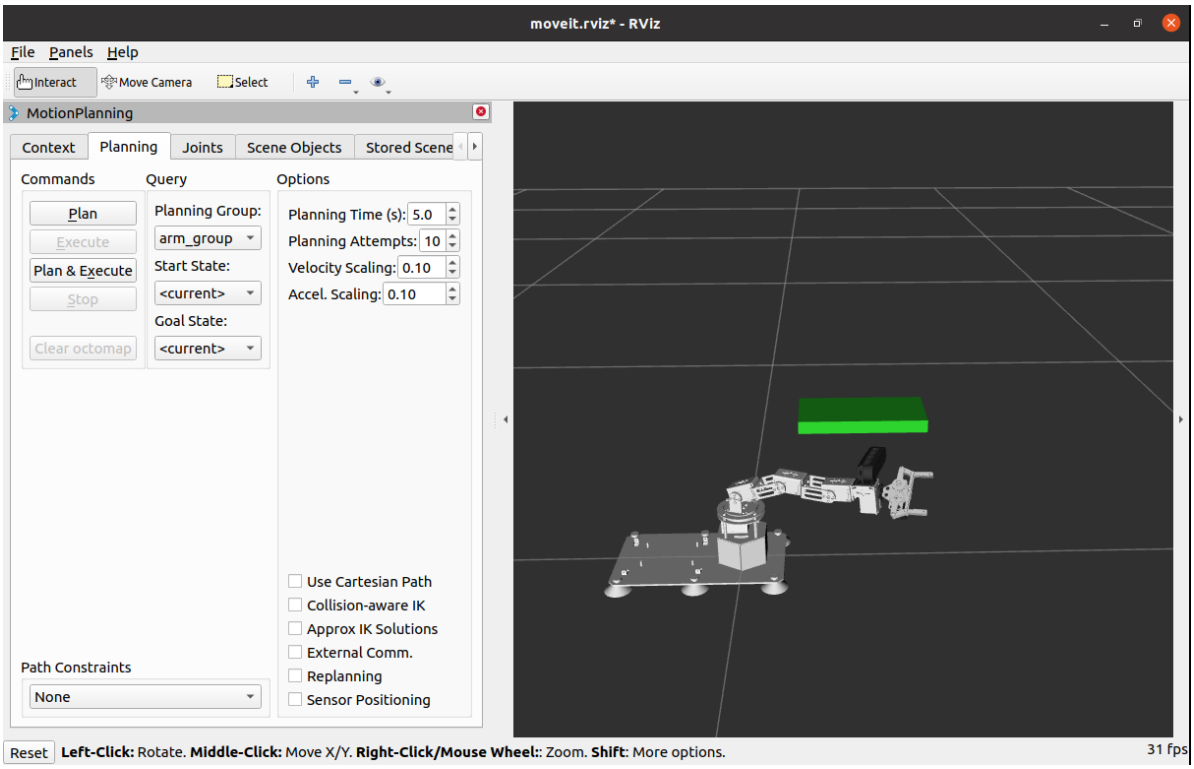
2. Start

Enter the following command in the terminal to start,

```
#Start MoveIt
roslaunch dofbot_pro_config demo.launch
```

After MoveIt is successfully started, enter in the terminal,

```
#Start the collision detection program
rosrun arm_moveit_demo 05_attached_object.py
```



After the program is started, add a board to Rviz, and the robot arm in rviz will randomly plan the movement, and will avoid the added board during the period.

3. Core code analysis

Code path:
/home/jetson/dofbot_pro_ws/src/arm_moveit_demo/scripts/05_attached_object.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy, sys
from time import sleep
import moveit_commander
from geometry_msgs.msg import PoseStamped
from moveit_commander import MoveGroupCommander, PlanningSceneInterface

if __name__ == "__main__":
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('attached_object_py')
    scene = PlanningSceneInterface()
    dofbot_pro = MoveGroupCommander('arm_group')
    # When motion planning fails, allow re-planning
    dofbot_pro.allow_replanning(True)
    dofbot_pro.set_planning_time(5)
    dofbot_pro.set_num_planning_attempts(10)
    dofbot_pro.set_goal_position_tolerance(0.01)
    dofbot_pro.set_goal_orientation_tolerance(0.01)
    dofbot_pro.set_goal_tolerance(0.01)
    dofbot_pro.set_max_velocity_scaling_factor(1.0)
    dofbot_pro.set_max_acceleration_scaling_factor(1.0)
    dofbot_pro.set_named_target("up")
    dofbot_pro.go()
    sleep(0.5)
    # Set the three-dimensional size of the obstacle [length, width and height]
    table_size = [0.1, 0.2, 0.02]
    table_pose = PoseStamped()
    table_pose.header.frame_id = 'base_link'
    table_pose.pose.position.x = 0.0
    table_pose.pose.position.y = 0.2
    table_pose.pose.position.z = 0.25 + table_size[2] / 2.0
    table_pose.pose.orientation.w = 1.0
    scene.add_box('obj', table_pose, table_size)
    rospy.sleep(2)
    dofbot_pro.set_named_target("down")
    dofbot_pro.go()
    sleep(0.5)
    index = 0
    while index < 10:
        # Set random target point
        dofbot_pro.set_random_target()
        # Start exercising
        dofbot_pro.go()
        sleep(0.5)
        index += 1
        print ("第 {} 次规划!!!".format(index))
    scene.remove_attached_object('obj')
```

```
    scene.remove_world_object()
    moveit_commander.roscpp_shutdown()
    moveit_commander.os._exit(0)
```