RealMan RM65-B Robot ROS Manual V1.0

RealMan (Beijing) Intelligent Technology Co., Ltd.

Version History

Version	Date	Comment
V1.0	2021-8-18	Init

1. Package introduction

#	Name	Functionality
1	rm_65_description	RM65-B robot functionality description, which contains robot models and configuration files, rm_65.urdf.xacro: RM65-B (no gripper) configuration file.
2	rm_65_moveit_config	To use Setup Assistant tool to create a MoveIt! Configuration file by robot URDF model, rm_65.urdf.xacro. It contains most required configuration files and launch files when MoveIt! runs, as well as a simple demo.
3	rm_gazebo	gazebo simulation robot parameters and file configuration.
4	rm_65_demo	MoveIt! Code demo, including scene planning, obstacle avoidance planning, and pick and place.
5	rm_msgs	All the control and status messages used by RM65-B
6	rm_control	The robot controller subdivides the robot arm trajectory planned by Moveit through cubic spline interpolation, and sends it to the rm_driver node according to the control cycle of 20ms. The cycle is adjustable, but it should be larger than 10ms.
7	rm_driver	（1）To establish a socket connection with the robot through the Ethernet port. Robot IP：192.168.1.18. Please ensure that the IP of the host computer is in the same local network. Use ROS to control the robot. Make sure that the robot is in Ethernet port communication mode. ； （2）Subscribe to each topic data, update the angle of each joint of the robot in RVIZ
8	rm_bringup	After starting the robot and running the corresponding launch file, it can automatically run rm_driver, rm_control and moveit interactive RVIZ interface. Drag the robot directly in the simulation interface to control the real robot movement.

2. Environmental requirements

■ OS：Ubuntu 18.04

■ ROS：melodic

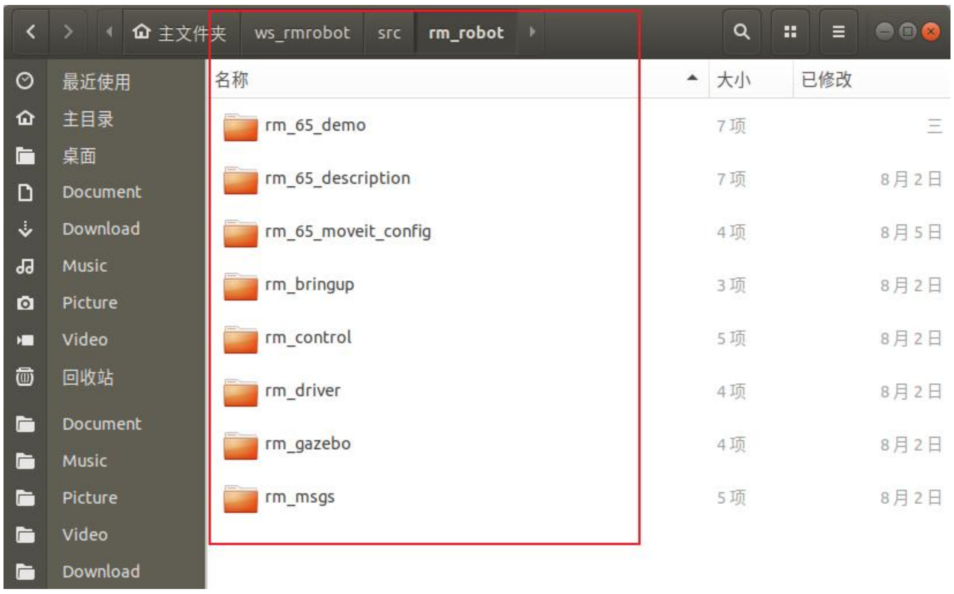
■ Others：Moveit! is installed；Gazebo is ready-to-use；ros_control plug-ins are ready-to-use.

3. Source code installation and compilation

To create a workspace named ws_rmrobot, use the following commands.

```
mkdir -p ~/ws_rmrobot/src
cd ~/ws_rmrobot/src/
```

Then copy the provided rm_robot source code to the src directory of the ws_rmrobot workspace or copy the source code to the src directory of other workspaces created by customer.



名称	大小	已修改
rm_65_demo	7 项	三
rm_65_description	7 项	8月2日
rm_65_moveit_config	4 项	8月5日
rm_bringup	3 项	8月2日
rm_control	5 项	8月2日
rm_driver	4 项	8月2日
rm_gazebo	4 项	8月2日
rm_msgs	5 项	8月2日

Fig. 3-1 Copying rm_robot source code to ws_rmrobot/src.

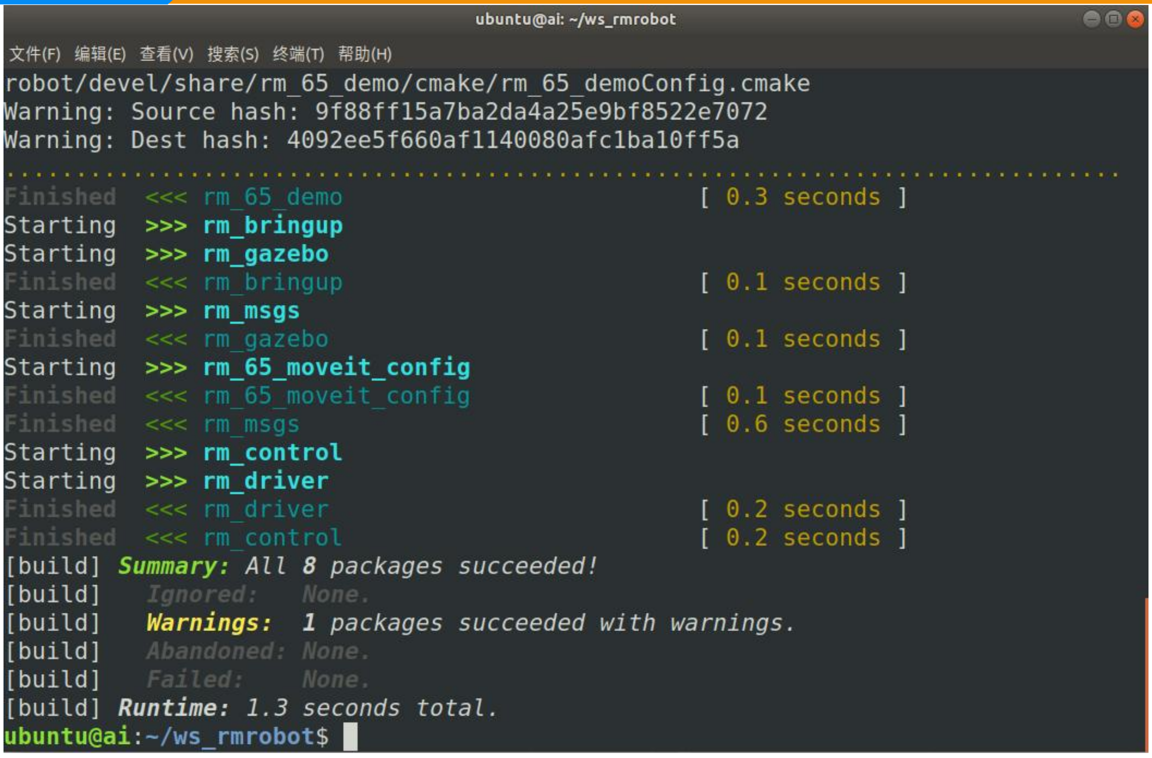
Then use rosdep to install dependent libraries as follows.

```
rosdep install -y --from-paths . --ignore-src --rosdistro melodic -r
```

Use the catkin tool to configure and compile the workspace as follows.

```
cd ~/ws_rmrobot
catkin init
catkin build rm_msgs
catkin build
```

Compilation is completed as shown in the figure below.



```
robot/devel/share/rm_65_demo/cmake/rm_65_demoConfig.cmake
Warning: Source hash: 9f88ff15a7ba2da4a25e9bf8522e7072
Warning: Dest hash: 4092ee5f660af1140080afc1ba10ff5a
..............................................................................
Finished  <<< rm_65_demo                          [ 0.3 seconds ]
Starting  >>> rm_bringup
Starting  >>> rm_gazebo
Finished  <<< rm_bringup                          [ 0.1 seconds ]
Starting  >>> rm_msgs
Finished  <<< rm_gazebo                           [ 0.1 seconds ]
Starting  >>> rm_65_moveit_config
Finished  <<< rm_65_moveit_config                 [ 0.1 seconds ]
Finished  <<< rm_msgs                             [ 0.6 seconds ]
Starting  >>> rm_control
Starting  >>> rm_driver
Finished  <<< rm_driver                           [ 0.2 seconds ]
Finished  <<< rm_control                          [ 0.2 seconds ]
[build] Summary: All 8 packages succeeded!
[build]   Ignored:   None.
[build]   Warnings:  1 packages succeeded with warnings.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 1.3 seconds total.
ubuntu@ai:~/ws_rmrobot$ 
```

Fig. 3-2 rm_robot source code successfully compiled.

4. Visualize robot arm model in rviz

4.1 Robot functionality description library

The rm_65_description function is included in the rm_robot source package, including the created robot model and configuration file.

It contains urdf, meshes, launch and config four folders.

- urdf: to store robot models in URDF/xacro files.
- meshes: to place model rendering files referenced in URDF.
- launch: to store related launch files.
- config: to save the configuration files of rviz.

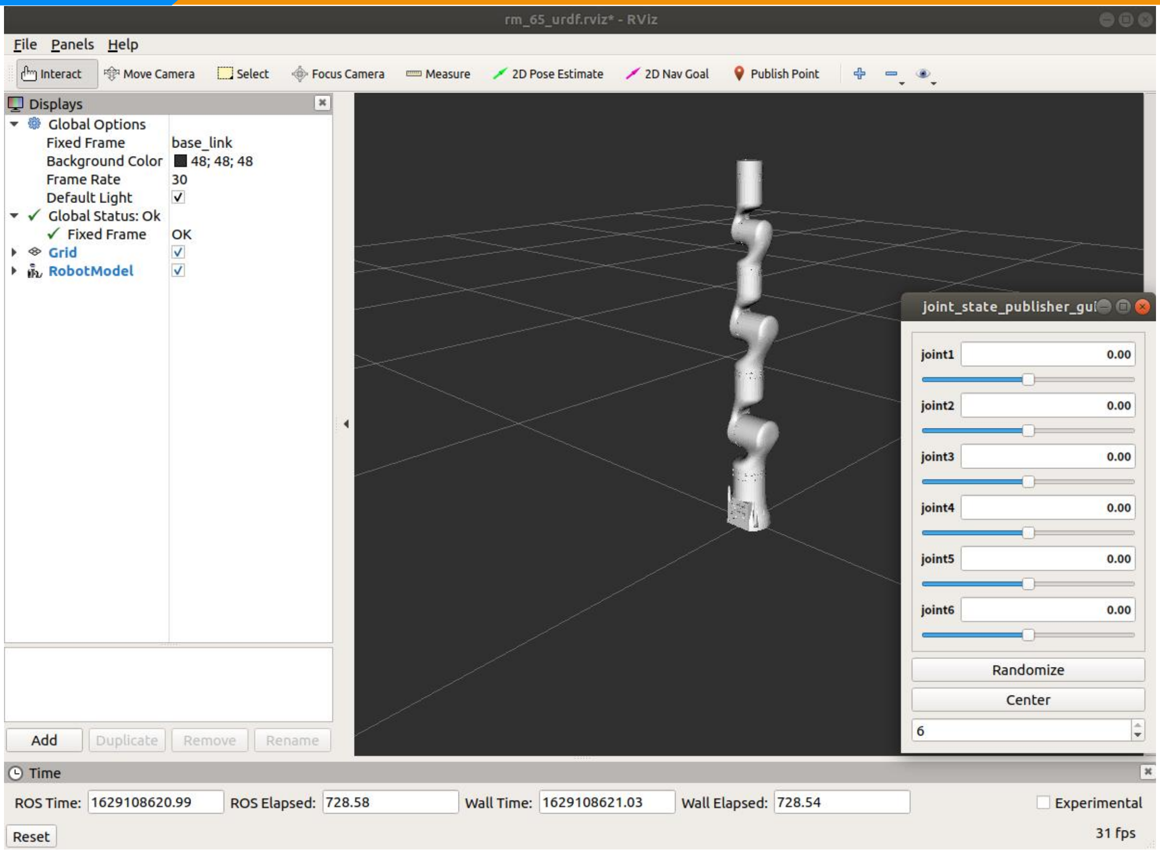
4.2 Visualize the models in rviz

The launch file used to visualize the rm_65 model has been created in the launch folder of the rm_65_description package, i.e., rm_65_description/launch/display_rm65.launch.

Open the terminal and enter the workspace and execute the following command to run the launch file.

```
cd ~/ws_rmrobot
source devel/setup.bash
roslaunch rm_65_description display_rm65.launch
```

Normally you can see the robot model shown in Fig.4-1 in rviz.



```
Global Options
  Fixed Frame        base_link
  Background Color   48; 48; 48
  Frame Rate         30
  Default Light      ✓
✓ Global Status: Ok
  ✓ Fixed Frame      OK
Grid                 ✓
RobotModel           ✓
```

```
joint1  0.00
joint2  0.00
joint3  0.00
joint4  0.00
joint5  0.00
joint6  0.00
6
```

```
ROS Time: 1629108620.99   ROS Elapsed: 728.58   Wall Time: 1629108621.03   Wall Elapsed: 728.54
31 fps
```

Fig. 4-1 RM65-B robot arm model in rviz.

After successful running, not only started rviz, but also a UI named "joint_state_publisher" appeared. This is because we started the joint_state_publisher node in the launch file, which can broadcast the state of each joint (except for the fixed type), and can also control the joint through the UI. Slide the control bar with the mouse in the control interface, and the corresponding robot arm joint in rviz will rotate.

If the model is not displayed in rviz, customer needs to manually modify the "Fixed Frame" as to "base_link", and then click the Add button at the bottom left to add "RobotModel" in the pop-up interface, as shown in Fig.4-2~4-3.

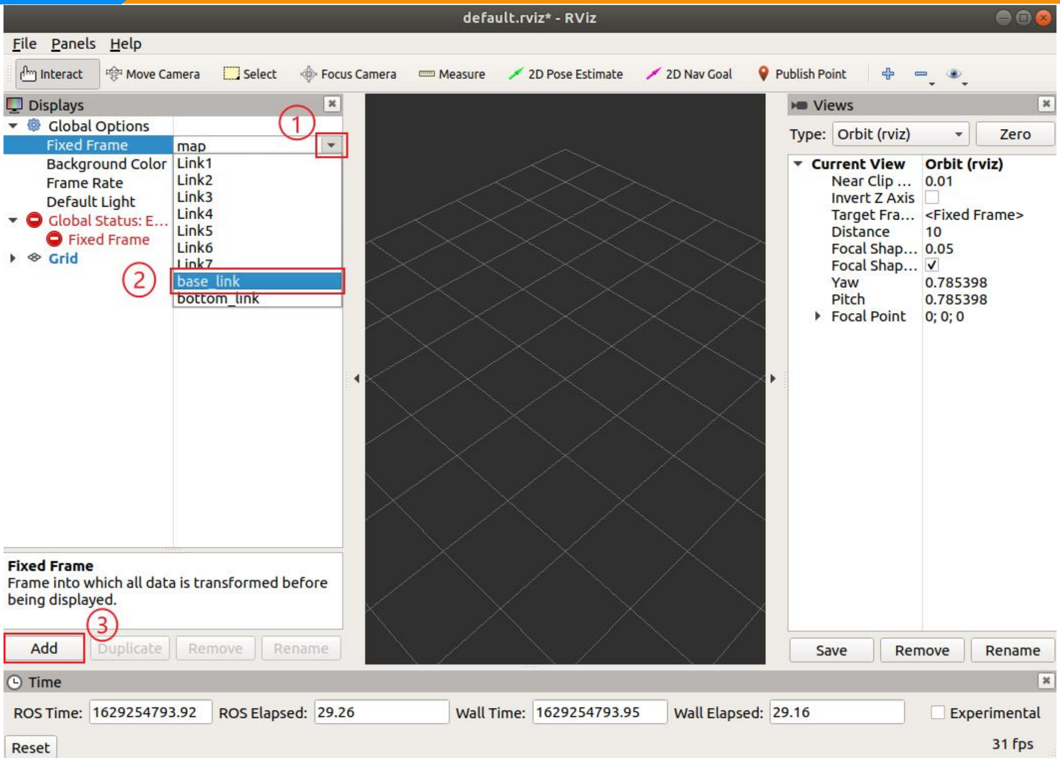


Fig. 4-2 Selecting base_link at the drop down box in FixedFrame.

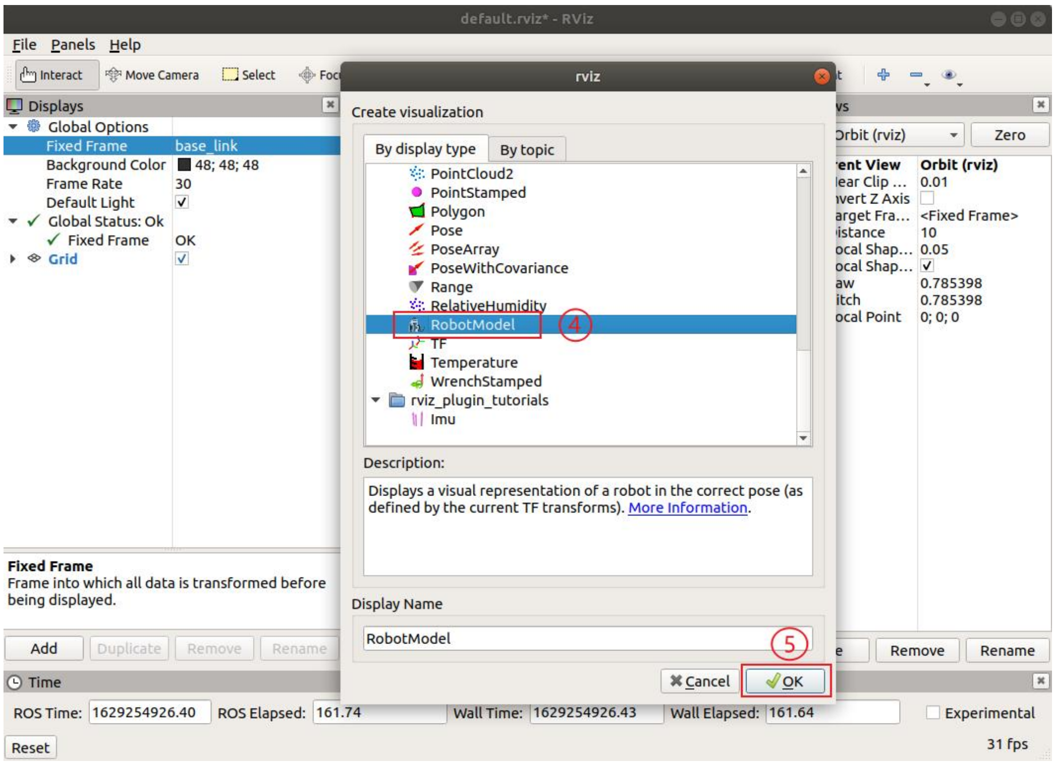


Fig. 4-3 Adding RobotModel in rviz.

5. Run MoveIt!

5.1 MoveIt! introduction

The motion planning of the robot arm is the most important part in the realization of autonomous grasping, which includes kinematics forward and inverse calculation, collision detection, environment perception, motion planning etc. The common motion planning of robot mostly uses the MoveIt! planning provided by the ROS system.

MoveIt! is a motion planning library that integrates component packages related to mobile operations in the ROS system. It contains most of the functions required in motion planning, and it provides a user-friendly configuration and debugging interface to facilitate the initialization and debugging of the robot in the ROS system.

Official website：http://moveit.ros.org/, where you can find MoveIt! tutorial and APIs.

5.2 Install MoveIt!

MoveIt! needs to be installed before use. If not installed, please use the following command to install it.

```
sudo apt install ros-melodic-moveit
sudo apt install ros-melodic-moveit-*
```

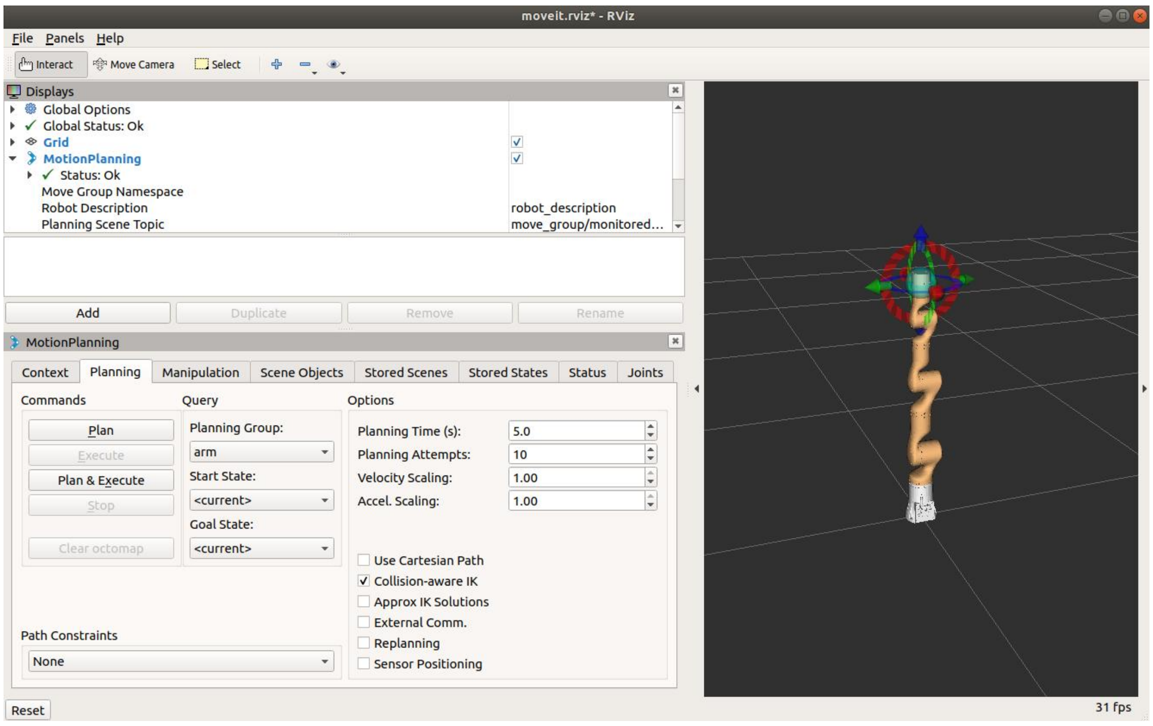
5.3 Run RM65-B MoveIt! demo

The rm_robot source code contains the rm_65_moveit_config package, which is a MoveIt! configuration package created by the Setup Assistant tool based on the URDF model of the robot arm. It contains most of the configuration and launch files required by MoveIt! It also contains a simple demo.

Open the terminal and enter the workspace and execute the following command to run the MoveIt! demo of the RM65-B robotic arm.

```
cd ~/ws_rmrobot
source devel/setup.bash
roslaunch rm_65_moveit_config demo.launch
```

After successful run, you can see the interface as shown in Fig. 5-1.


moveit.rviz* - RViz
File Panels Help
Interact Move Camera Select
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Add Duplicate Remove Rename
MotionPlanning
Context Planning Manipulation Scene Objects Stored Scenes Stored States Status Joints
Commands Query Options
Plan Execute Plan & Execute Stop Clear octomap
Path Constraints
Reset
31 fps

Move Group Namespace
Robot Description: robot_description
Planning Scene Topic: move_group/monitored...

Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00

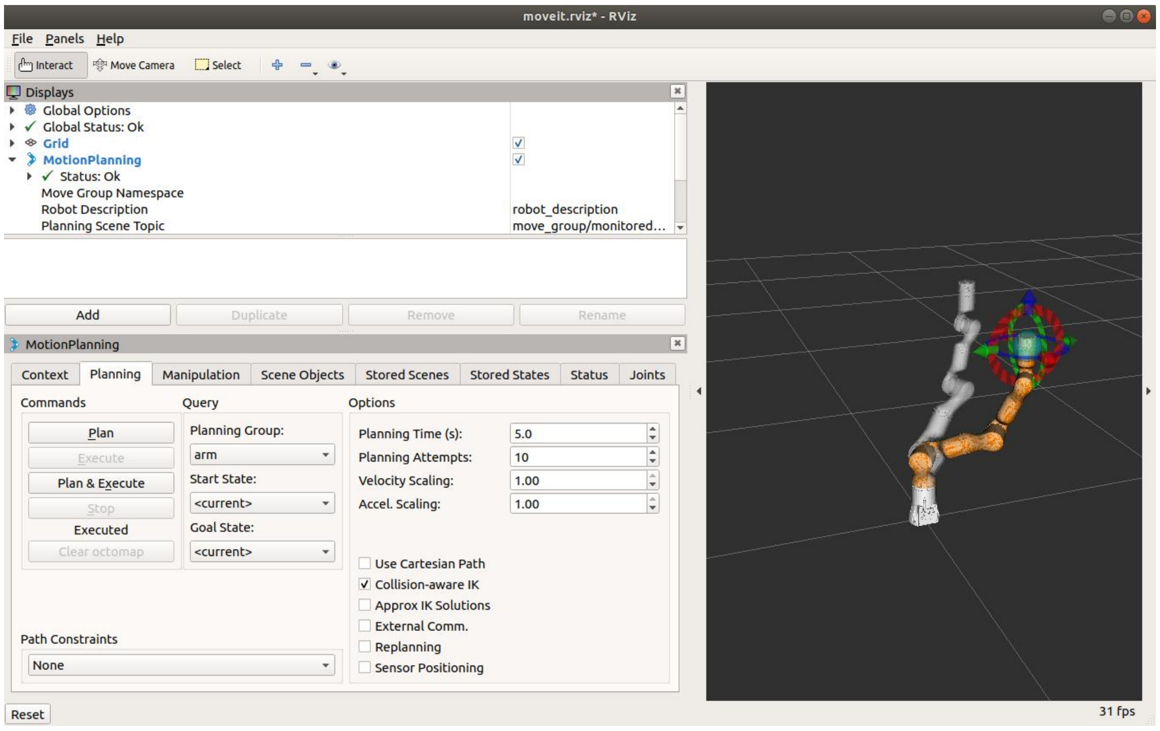
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 5-1 The start interface of RM65-B MoveIt! demo.

This interface adds MoveIt! plug-in based on rviz. The MoveIt! related functions can be configured through the plug-in window in the lower left corner to control the robot arm to complete motion planning. For example, through the MoveIt! plug-in, you can control the robot arm to complete functions such as drag planning, random target planning, initial position and pose update, and collision detection.

5.3.1 Drag planning

Drag the front end of the robot arm to change its position and pose. Then click the "Plan & Execute" button on the Planning tab, MoveIt! starts to plan the path and control the robot to move to the target position. From the right interface, you can see the entire process of the robot's movement (see Fig. 5-2).


moveit.rviz* - RViz
File Panels Help
Interact Move Camera Select
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Manipulation Scene Objects Stored Scenes Stored States Status Joints
Commands Query Options
Plan Execute Plan & Execute Stop Clear octomap
Reset
31 fps

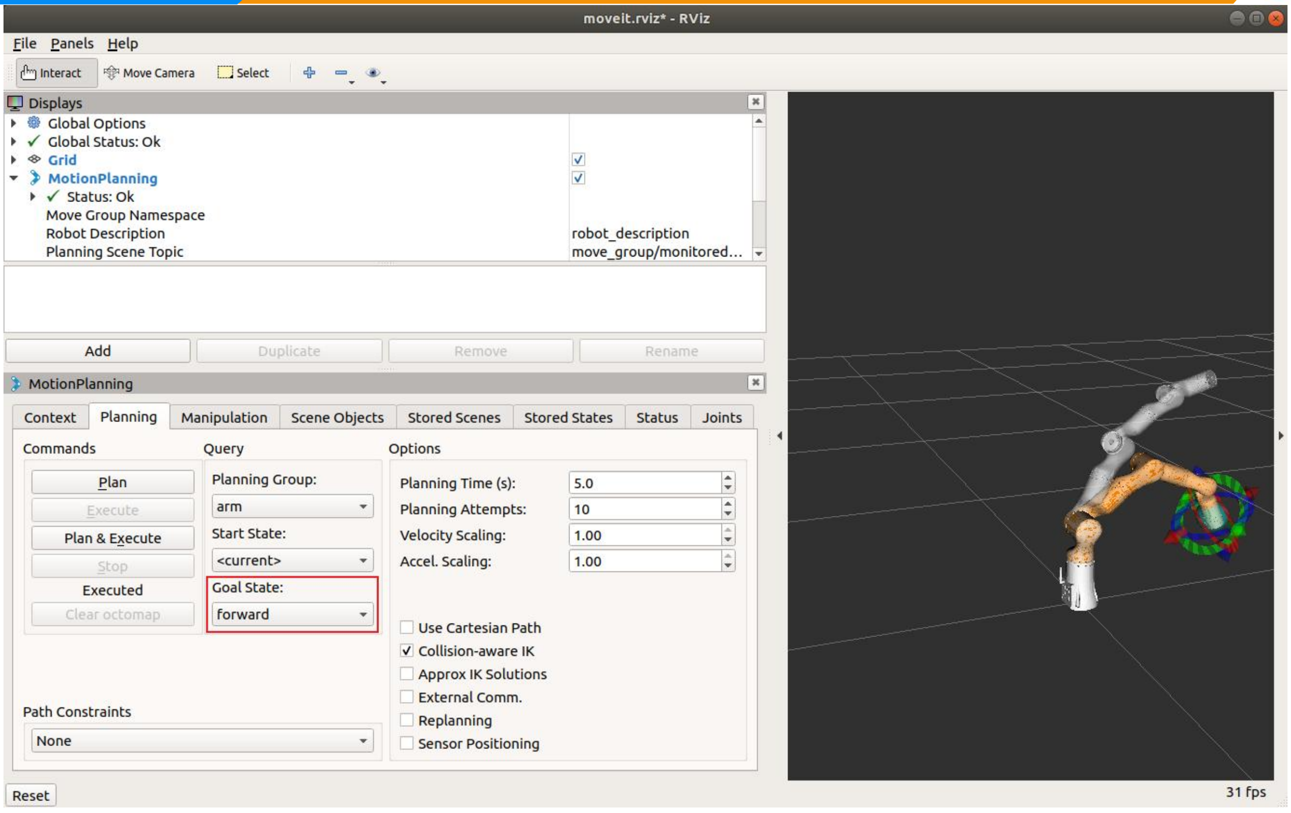
Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description: robot_description
Planning Scene Topic: move_group/monitored…

Executed
Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 5-2 Drag planning interface.

5.3.2 Select target motion planning

Click the Goal State drop-down list in the Planning tab to select the target position and pose of the robot arm, and then click the "Plan & Execute" button, MoveIt! starts to plan the path and control the robot to move to the target. You can see the robot from the right interface the whole process of movement (see Fig. 5-3).



Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description robot_description
Planning Scene Topic move_group/monitored...

Planning Group: arm
Start State: <current>
Goal State: forward

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00

☐ Use Cartesian Path
✓ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Path Constraints: None

Fig. 5-3 Target motion planning selection interface.

6. Use MoveIt! to control robot arm in Gazebo

6.1 Overview

The main idea of the joint simulation of MoveIt! and Gazebo is to build a bridge between ros_control and MoveIt!. First configure the joint and sensor interface yaml file by the MoveIt! and load it to the rviz; then configure the ros_control and interface yaml file on the robot to load it to Gazebo. Finally, start Gazebo (with ros_control) and rviz (with MoveIt) at the same time to achieve the purpose of joint simulation.

6.2 Configuration instructions

6.2.1 MoveIt! configuration

1） Controller configuration file: controllers_gazebo.yaml

controllers_gazebo.yaml is at rm_65_moveit_config/config as follows.

```
controller_manager_ns: controller_manager
controller_list:
  - name: arm/arm_joint_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - joint1
      - joint2
```

```
        - joint3
        - joint4
        - joint5
        - joint6
```

2）Modify the launch file: rm_65_moveit_controller_manager.launch.xml

Modify the rm_65_moveit_controller_manager.launch.xml in the rm_65_moveit_config package to load the controllers_gazebo.yaml file just created to the parameter server as follows.

```
<launch>
<arg name="execution_type" default="FollowJointTrajectory" />

<!-- loads moveit_controller_manager on the parameter server which is taken as argument
if no argument is passed, moveit_simple_controller_manager will be set -->
<arg                                                                name="moveit_controller_manager"
default="moveit_simple_controller_manager/MoveItSimpleControllerManager" />
<param name="moveit_controller_manager" value="$(arg moveit_controller_manager)"/>

<!-- load controller_list -->
<arg name="use_controller_manager" default="true" />
<param name="use_controller_manager" value="$(arg use_controller_manager)" />

<!-- loads ros_controllers to the param server -->
<!-- <rosparam file="$(find rm_65_moveit_config)/config/ros_controllers.yaml"/> -->
<!-- <rosparam file="$(find rm_65_moveit_config)/config/controllers.yaml"/> -->
<rosparam file="$(find rm_65_moveit_config)/config/controllers_gazebo.yaml"/>
</launch>
```

3）Run file moveit_planning_execution.launch

moveit_planning_execution.launch is a newly created launch file in the rm_65_moveit_config/launch directory to load MoveIt and rviz, and finally run the joint state broadcaster to issue instructions to Gazebo. Its source code is shown as follows.

```
<launch>
 # 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="$(find rm_65_moveit_config)/launch/move_group.launch">
  <arg name="publish_monitored_planning_scene" value="true" />
 </include>
 # The visualization component of MoveIt!
 <include file="$(find rm_65_moveit_config)/launch/moveit_rviz.launch"/>

  <!-- We do not have a robot connected, so publish fake joint states -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam param="/source_list">[/arm/joint_states]</rosparam>
    </node>
</launch>
```

6.2.2 Robot end configuration

➢ Joint trajectory controller

The output interface of MoveIt! after completing the motion planning is an action named "FollowJointTrajectory", which contains a series of planned path point trajectories. This information can be converted to the joint position required by the robot in Gazebo by a "Joint Trajectory Controller" provided by ros_control.

1）Configuration file: rm_65_trajectory_control.yaml

The use of the controller “FollowJointTrajectory” first requires a configuration file to configure the control parameters for the trajectory of the six axes of the robot arm, that is, the rm_65_trajectory_control.yaml configuration file in the rm_gazebo/config directory. The code is as follows.

```
arm:
  arm_joint_controller:
    type: "position_controllers/JointTrajectoryController"
    joints:
      - joint1
      - joint2
      - joint3
      - joint4
      - joint5
      - joint6

    gains:
      joint1:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint2:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint3:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint4:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint5:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
      joint6:    {p: 1000.0, i: 0.0, d: 0.1, i_clamp: 0.0}
```

2）Controller launch file

To load Joint Trajectory Controller, use arm_65_trajectory_controller.launch file in the rm_gazebo/launch directory.

```
<launch>
    <rosparam file="$(find rm_gazebo)/config/rm_65_trajectory_control.yaml" command="load"/>
    <node name="arm_controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
          output="screen" ns="/arm" args="arm_joint_controller"/>
</launch>
```

➢ Joint state controller

The joint state controller is an optional plug-in. Its main function is to broadcast the joint state and TF transformation of the robot. Otherwise, you cannot see the coordinate system option in the drop-down list in the Fixed Frame setting of rviz. You can only input it manually, but it can still be normally used.

1）Configuration file of joint state controller: arm_gazebo_joint_states.yaml

arm_gazebo_joint_states.yaml file is the rm_gazebo/config directory as follows.

```
arm:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
```

2）Launch file arm_gazebo_states.launch to load paramers

arm_gazebo_states.launch file is the rm_gazebo/launch directory.

```
<launch>
    <!-- Load the configuration parameters of the joint controller into the parameter server -->
    <rosparam file="$(find rm_gazebo)/config/arm_gazebo_joint_states.yaml" command="load"/>

    <node name="joint_controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
          output="screen" ns="/arm" args="joint_state_controller" />

    <!—Run robot_state_publisher node, broadcast tf   -->
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
        respawn="false" output="screen">
        <remap from="/joint_states" to="/arm/joint_states" />
    </node>
</launch>
```

6.2.3 Launch file: arm_65_bringup_moveit.launch

The top-level launch file arm_65_bringup_moveit.launch is in the rm_gazebo/launch directory to start Gazebo, load all the controllers, and finally start MoveIt!. Its source code is shown as follows.

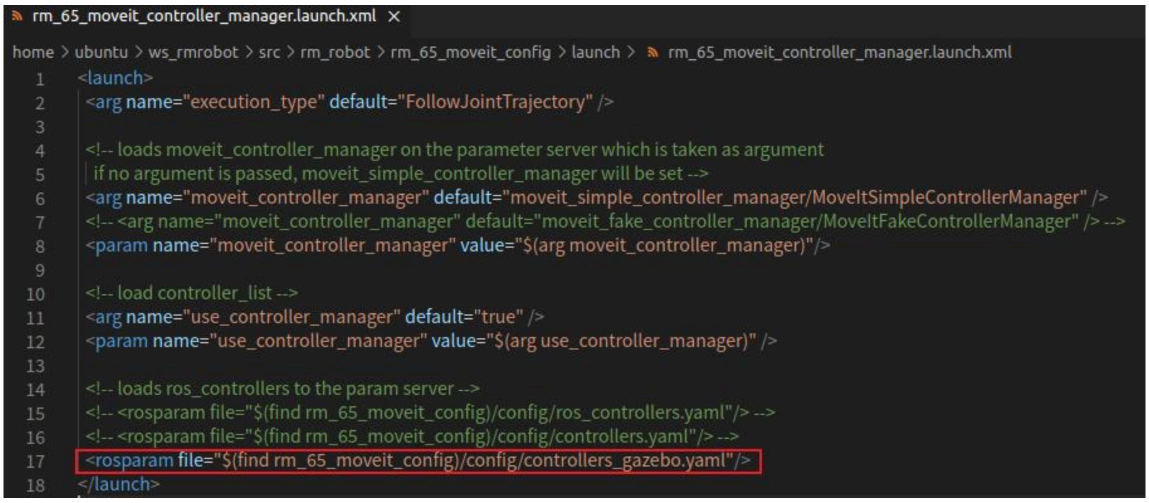
```
<launch>
    <!-- Launch Gazebo   -->
    <include file="$(find rm_gazebo)/launch/arm_world.launch" />

    <!-- ros_control arm launch file -->
    <include file="$(find rm_gazebo)/launch/arm_gazebo_states.launch" />

    <!-- ros_control trajectory control dof arm launch file -->
    <include file="$(find rm_gazebo)/launch/arm_65_trajectory_controller.launch" />
    <!-- moveit launch file -->
    <include file="$(find rm_65_moveit_config)/launch/moveit_planning_execution.launch" />
</launch>
```

6.3 Running results

Before starting the node, you need to ensure that the configuration in the rm_65_moveit_config/launch/rm_65_moveit_controller_manager.launch.xml file is the controllers_gazebo.yaml file that is loaded to the parameter server. See Fig.6-1.



```
rm_65_moveit_controller_manager.launch.xml ×
home > ubuntu > ws_rmrobot > src > rm_robot > rm_65_moveit_config > launch > rm_65_moveit_controller_manager.launch.xml
<launch>
  <arg name="execution_type" default="FollowJointTrajectory" />

  <!-- loads moveit_controller_manager on the parameter server which is taken as argument
   if no argument is passed, moveit_simple_controller_manager will be set -->
  <arg name="moveit_controller_manager" default="moveit_simple_controller_manager/MoveItSimpleControllerManager" />
  <!-- <arg name="moveit_controller_manager" default="moveit_fake_controller_manager/MoveItFakeControllerManager" /> -->
  <param name="moveit_controller_manager" value="$(arg moveit_controller_manager)"/>

  <!-- load controller_list -->
  <arg name="use_controller_manager" default="true" />
  <param name="use_controller_manager" value="$(arg use_controller_manager)" />

  <!-- loads ros_controllers to the param server -->
  <!-- <rosparam file="$(find rm_65_moveit_config)/config/ros_controllers.yaml"/> -->
  <!-- <rosparam file="$(find rm_65_moveit_config)/config/controllers.yaml"/> -->
  <rosparam file="$(find rm_65_moveit_config)/config/controllers_gazebo.yaml"/>
</launch>
```

Fig. 6-1 rm_65_moveit_controller_manager.launch.xml to load controllers_gazebo.yaml

Run the following commands to start MoveIt! and Gazebo.

```
cd ~/ws_rmrobot
source devel/setup.bash
roslaunch rm_gazebo arm_65_bringup_moveit.launch
```

The interface of Gazebo is shown in Fig.6-2.

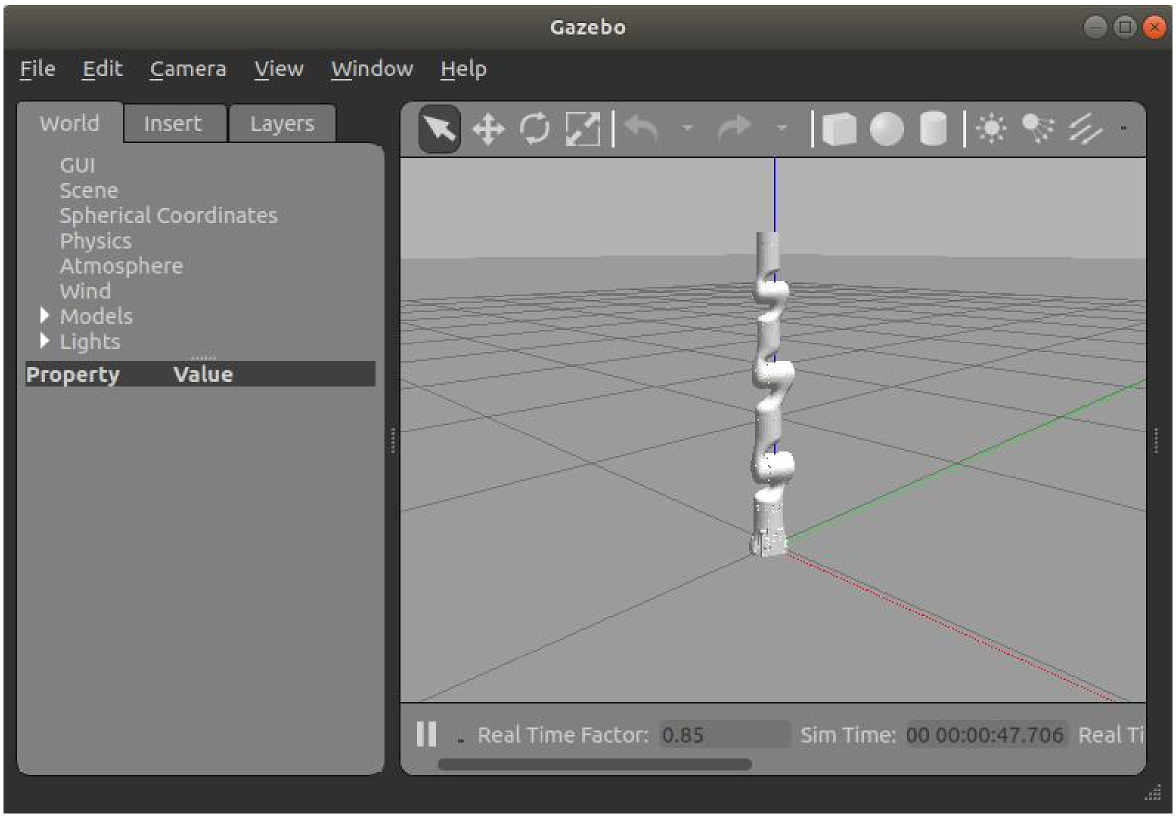
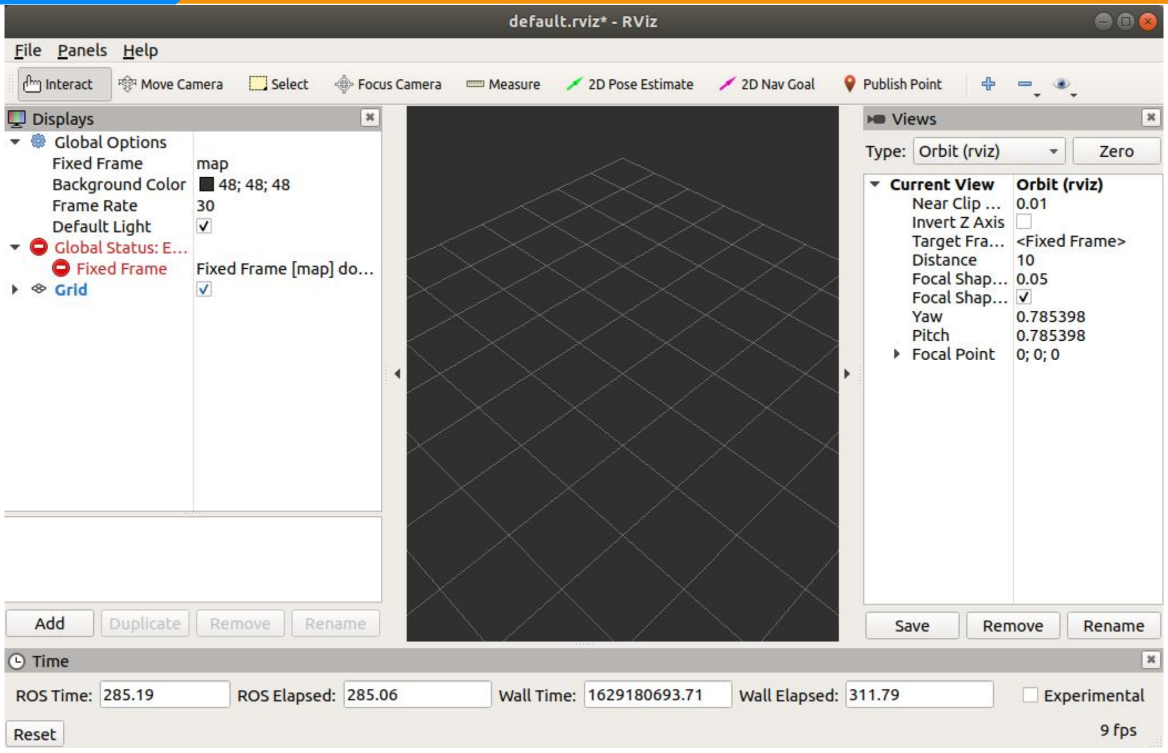


Fig. 6-2 RM65-B robot shown in Gazebo.

The interface of rvizis shown in Fig.6-3.



Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: E...
Fixed Frame Fixed Frame [map] do...
Grid ✓

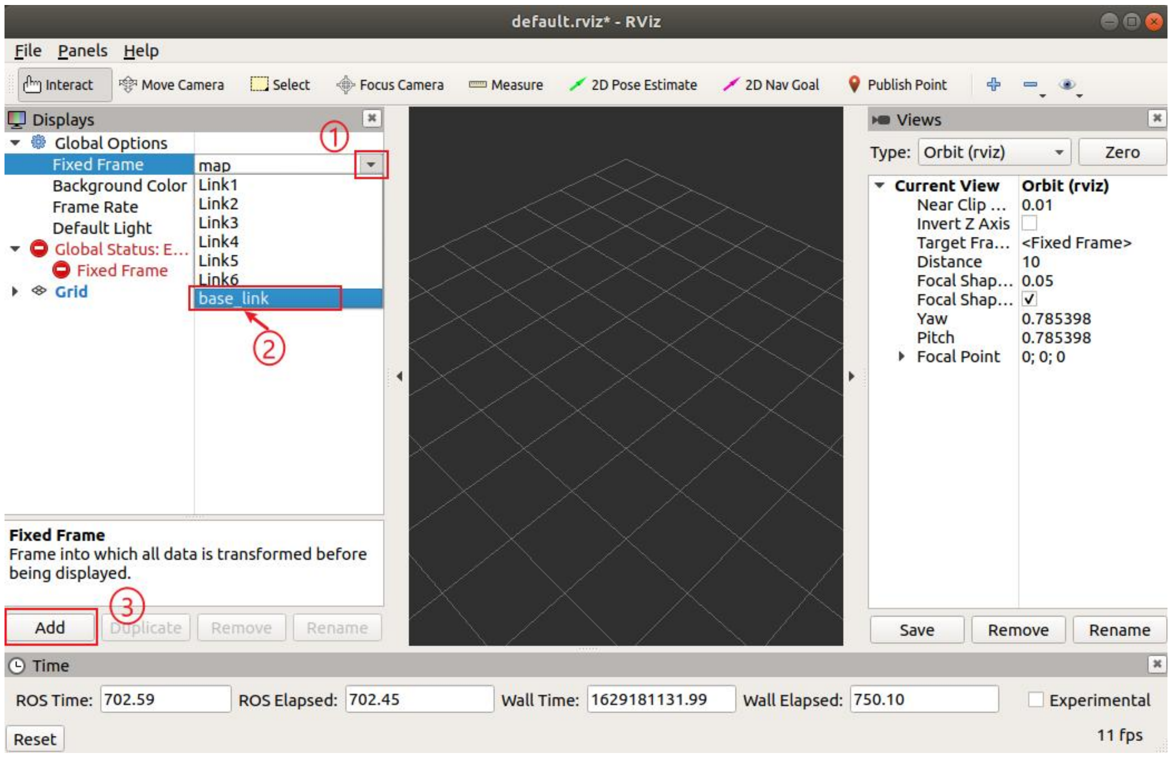
Type: Orbit (rviz)
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis
Target Fra... <Fixed Frame>
Distance 10
Focal Shap... 0.05
Focal Shap... ✓
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0

Time
ROS Time: 285.19 ROS Elapsed: 285.06 Wall Time: 1629180693.71 Wall Elapsed: 311.79 Experimental
9 fps

Fig.6-3 rviz interface.

In the opened rviz, no robot model can be seen for the time being, and some errors would even be prompted. Next, a simple configuration will solve these problems.

First modify the "Fixed Frame" to "base_link"; then click the "Add" button in the plug-in list column on the left to add MotionPlanning to the control window. At this point, you should see the robot appear in the main interface on the right, and the posture of the robot in Gazebo should be consistent with the display in rvize. The operation is shown in Fig. 6-4 ~ Fig. 6-6.



Global Options
Fixed Frame map
Background Color
Frame Rate
Default Light
Global Status: E...
Fixed Frame
Grid

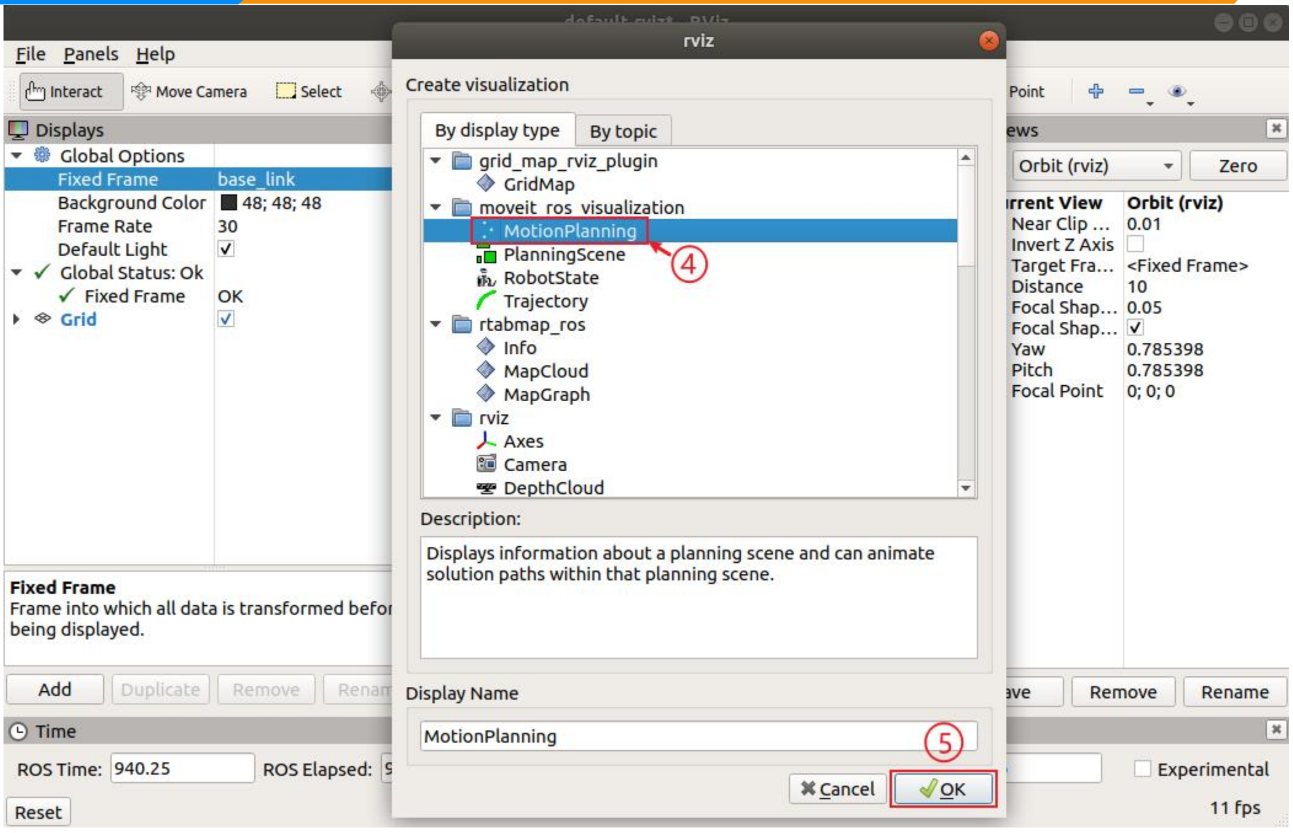
map
Link1
Link2
Link3
Link4
Link5
Link6
base_link

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis
Target Fra... <Fixed Frame>
Distance 10
Focal Shap... 0.05
Focal Shap... ✓
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0

Time
ROS Time: 702.59 ROS Elapsed: 702.45 Wall Time: 1629181131.99 Wall Elapsed: 750.10 Experimental
11 fps

Fig. 6-4 Modifying “Fixed Frame” to “base_link” in rviz.



Description:

Displays information about a planning scene and can animate solution paths within that planning scene.

Display Name: MotionPlanning

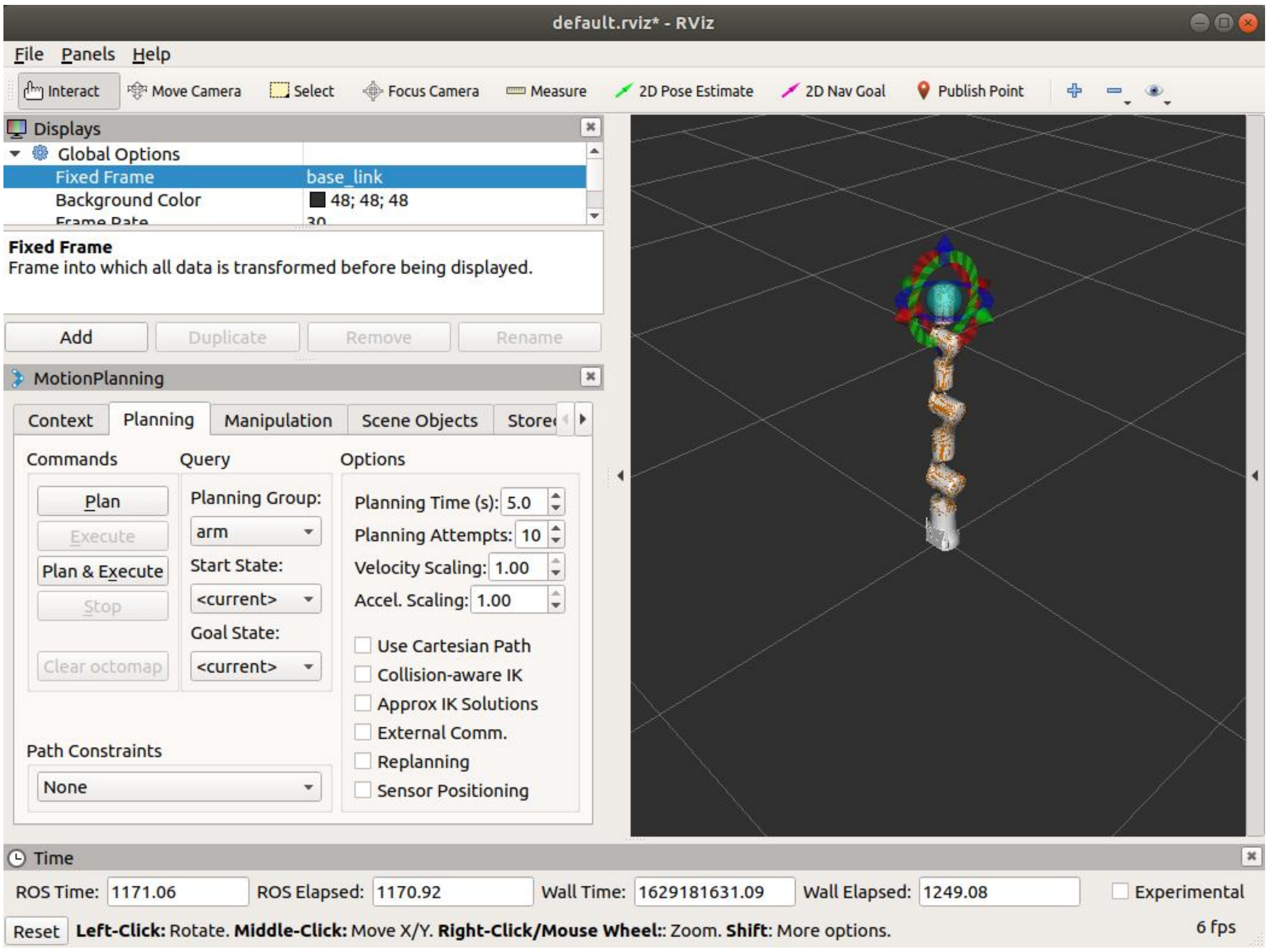
Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
Global Status: Ok
Fixed Frame: OK
Grid: ✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0.01
Invert Z Axis:
Target Fra...: <Fixed Frame>
Distance: 10
Focal Shap...: 0.05
Focal Shap...: ✓
Yaw: 0.785398
Pitch: 0.785398
Focal Point: 0; 0; 0

ROS Time: 940.25

Fig. 6-5 Adding MotionPlanning to the control window in rviz.



Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30

Fixed Frame
Frame into which all data is transformed before being displayed.

Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None

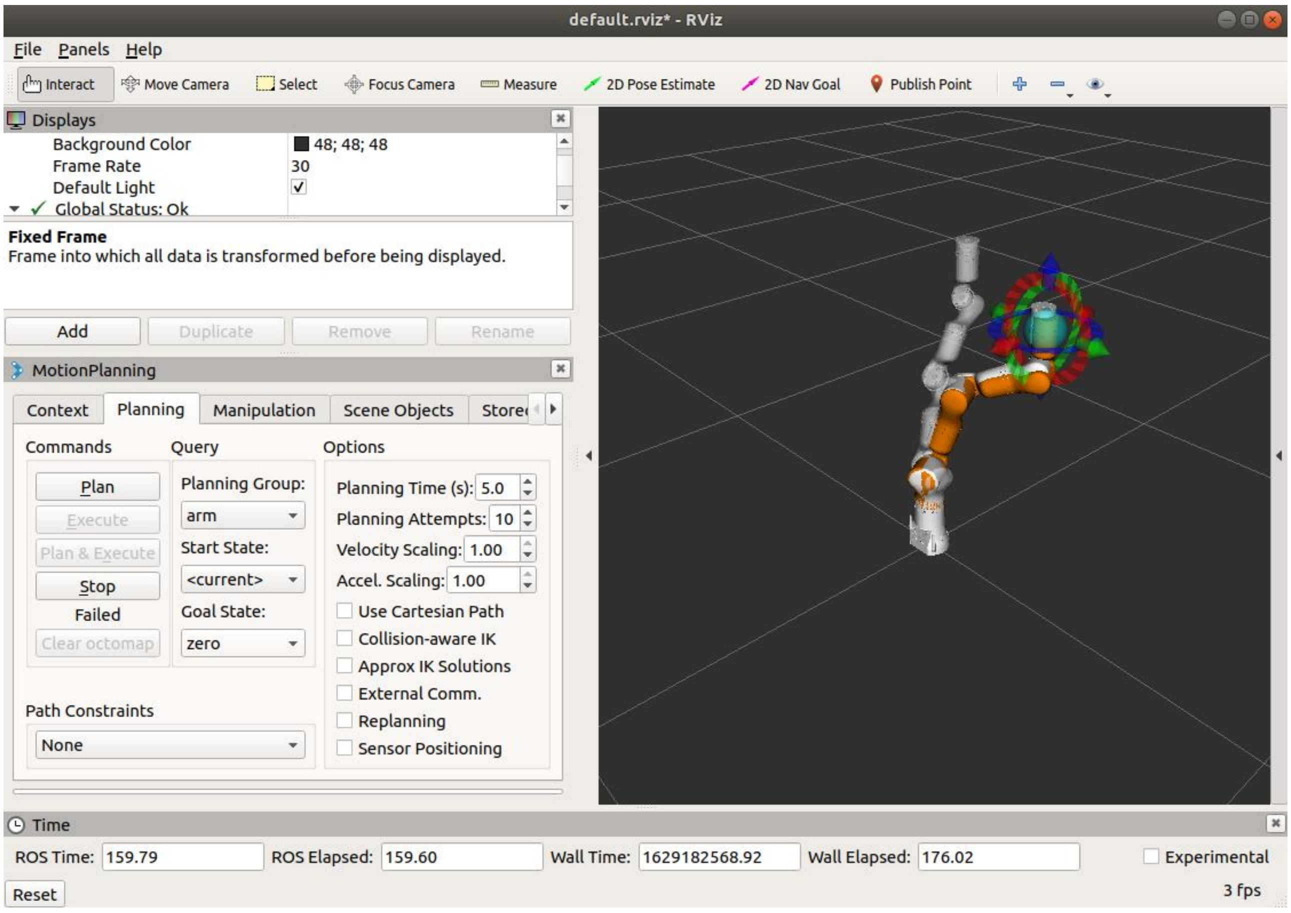
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning

ROS Time: 1171.06 ROS Elapsed: 1170.92 Wall Time: 1629181631.09 Wall Elapsed: 1249.08

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Fig. 6-6 RM65-B displayed in rviz.

Next, use MoveIt! to plan motion in several ways to control the robot in Gazebo. In Fig. 6-7, we drag the end of the robot to a position, and then click the "Plan & Execute" button, then you can see the robot in rviz starts to execute and you can see that the robot in Gazebo starts to move in a way that is consistent with the robot model in rviz, as shown in Fig.6-8.



Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
Global Status: Ok

Fixed Frame
Frame into which all data is transformed before being displayed.

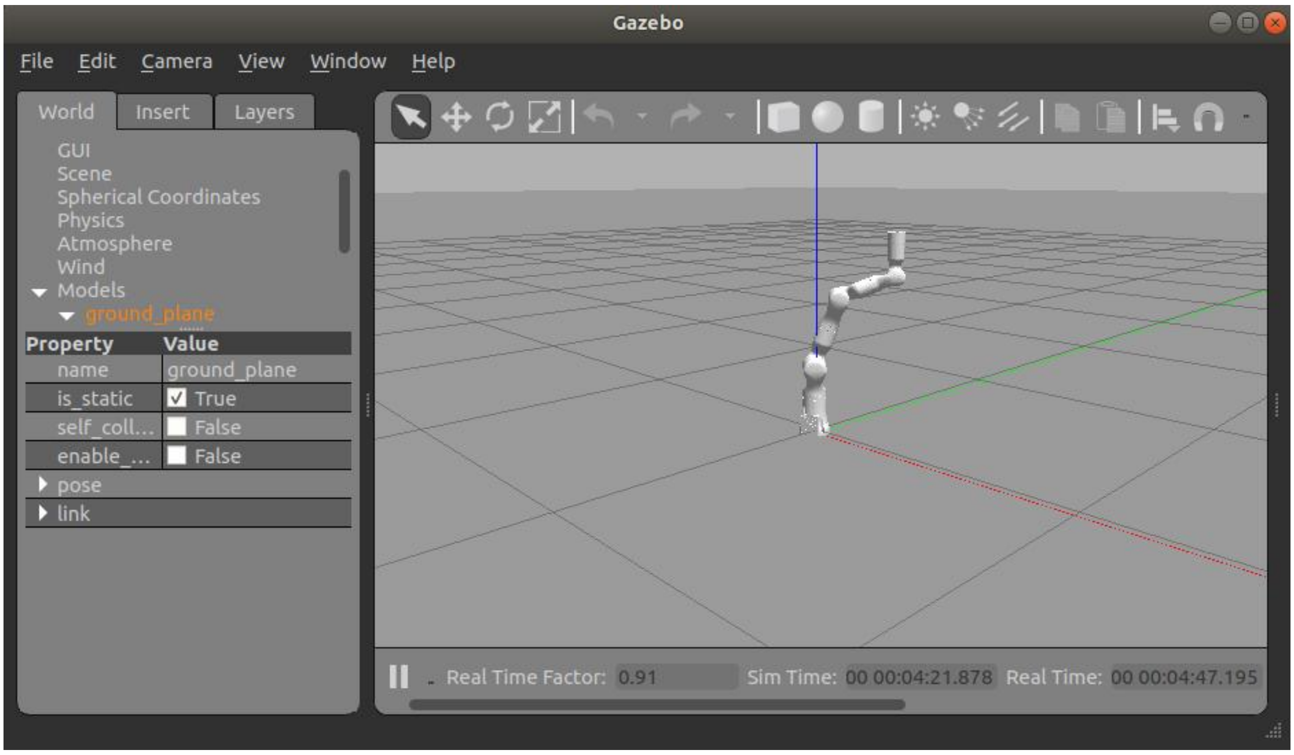
Failed

Planning Group: arm
Start State: <current>
Goal State: zero
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

ROS Time: 159.79 ROS Elapsed: 159.60 Wall Time: 1629182568.92 Wall Elapsed: 176.02 ☐ Experimental
3 fps

Fig. 6-7 Using MoveIt! to drag, plan and execute.



Property	Value
name	ground_plane
is_static	✓ True
self_coll...	False
enable_...	False
pose	
link	

Real Time Factor: 0.91 Sim Time: 00 00:04:21.878 Real Time: 00 00:04:47.195

Fig. 6-8 The robot in Gazebo executing synchronously according to the rviz.

7. Use MoveIt! to control a real robot arm

7.1 Overview

The output interface of MoveIt! after completing motion planning is an action named "FollowJointTrajectory", which contains a series of planned path point trajectories. Unlike using MoveIt! to control the robot in Gazebo, the virtual robot has the ros_control plug-in in gazebo which automatically helps us obtain the action information of follow_joint_trajectory. To the real robot, we need to add a server to subscribe to this action and process and then control the real robot.

7.1.1 Robot controller: rm_control

The robot controller rm_control package, by adding a server to subscribe to the action information output by MoveIt! after completing the motion planning, and then subdividing the robot arm trajectory planned by Moveit through cubic spline interpolation, and sending it to the rm_driver node according to the control cycle of 20ms. See the source code for specific implementation.

7.1.2 Robot drive package: rm_driver

The rm_driver package realizes the establishment of a socket connection with the robot through the Ethernet port, subscribes and broadcasts/publishes the topic information of the robot, and sends the path point trajectory of the robot processed by rm_control to the real robot arm through the socket realizing the control of the real robot. At the same time, it receives information from the robot and publishes it to move_group via topic to complete the synchronization of the robot in rviz. See the source code for specific implementation.

See the Table 7-1 for information about rm_driver and rm_control node subscriptions and broadcasting/publications.

Table 7-1 topic information of rm_driver and rm_control node subscriptions and broadcasting/publications

Node	Category	Topic Name	msg type	Functionality
rm_driver (Robot drive node)	subscribe	/rm_driver/MoveJ_Cmd	rm_msgs::MoveJ	Obtain the MoveJ planning instruction and send it to the robot.
		/rm_driver/MoveL_Cmd	rm_msgs::MoveL	Obtain the MoveL planning instruction and send it to the robot.
		/rm_driver/MoveC_Cmd	rm_msgs::MoveC	Obtain the MoveC planning instruction and send it to the robot.
		/rm_driver/JointPos	rm_msgs::JointPos	Obtain the angle passthrough command, which is not planned by the controller, and is directly sent to the joint to operate. The command and the current angle cannot exceed 30 degrees.
		/rm_driver/Arm_Digital_Output	rm_msgs::Arm_Digital_Output	Obtain the digital IO output control of the robot controller and send it to the robot.

		/rm_driver/Arm_Analogl_Output	rm_msgs::Arm_Analog_Output	Obtain the analog IO output control of the robot controller and send it to the robot.
		/rm_driver/Tool_Digital_Output	rm_msgs::Tool_Digital_Output	Obtain the digital IO output control of the end of the robot arm and send it to the robot.
		/rm_driver/Tool_Analogl_Output	rm_msgs::Tool_Analog_Output	Obtain the analog IO output control of the end of the robot arm and send it to the robot.
		/rm_driver/IO_Update	rm_msgs::IO_Update	Obtain the IO input update instruction and send it to the robot.
		/rm_driver/Gripper_Cmd	rm_msgs::Gripper	Obtain the control command of the gripper and send it to the robot.
		/rm_driver/Emergency_Stop	rm_msgs::Stop	Obtain the emergency stop command and send it to the robot.
	publish	/joint_states	sensor_msgs::JointState	The actual angle of the arm joint.
		/rm_driver/Arm_IO_State	rm_msgs::Arm_IO_State	Robot controller IO input state or status.
		/rm_driver/Tool_IO_State	rm_msgs::Tool_IO_State	IO input state at the end of the robot arm.
		/rm_driver/Plan_State	rm_msgs::Plan_State	Robot arm trajectory planning state.
rm_control (Robot controller)	publish	/rm_driver/JointPos	rm_msgs::JointPos	The moveit trajectory planning point is subdivided by cubic spline interpolation, and the subdivision period is 10ms. Then the interpolated trajectory point is sent to rm_driver according to the period to control the robot.

7.1.3 rm_msgs package

The rm_msgs package defines all control messages and state or status messages used by RM65-B robot. Both rm_control and rm_driver are required.

See the table for specific message types Table 7-2.

Table 7-2 Message types defined in rm_msgs package

Message type	Functionality	Composition	Type	Definition
Arm_Analog_Output	Controller analog IO output	num	uint8	Analog IO output channel number, 1~4
		voltage	float32	Output voltage, 0~10V

Arm_Digital_Output	Controller digital IO output	num	uint8	Digital IO output channel number, 1~4
		state	bool	Output state, false-low voltage, true-high voltage
Arm_IO_State	Controller input IO state	Arm_Digital_Input	bool[3]	Digital IO input state, channel 1~3
		Arm_Analog_Input	float32[4]	Analog IO input state, channel 1~4
Gripper_Pick	Gripper gripping control	speed	uint16	Gripping speed, range:1~1000
		force	uint16	Grip force, range:1~1000
JointPos	Joint angle	joint	float32[6]	6 joint angles
MoveC	Circular planning command	Mid_Pose	geometry_msgs/Pose	Middle point of circular planning
		End_Pose	geometry_msgs/Pose	End of circular planning
		speed	float32	Speed percentile，0~1
MoveJ	Joint space planning command	joint	float32[6]	Target position joint angle
		speed	float32	Speed percentile, 0~1
MoveL	Linear planning command	Pose	geometry_msgs/Pose	Target pose
		speed	float32	Speed percentile, 0~1
Plan_State	Trajectory planning return state	state	bool	false-Planning failed， true-Target reached.
Stop	Robot emergency stop command	state	bool	true-Robot emergency stop
Tool_Analog_Output	Analog IO output command at the tool end of the robot arm	voltage	float32	Analog IO output voltage, range: 0~10V
Tool_Digital_Output	Digital IO output command at the tool end of the robot arm	num	uint8	Digital IO output channel number, 1~2
		state	bool	Output state ， false-Low voltage， true-Hight voltage
Tool_IO_State	IO input state of at the tool end of the robot arm	Tool_Digital_Input	bool[2]	digital IO input state, channel 1~2
		Tool_Analog_Input	float32	Analog IO input state
Gripper_Set	Set the opening range of the gripper	position	uint16	Target position, range: 1~1000, representing position: 0~70mm
Joint_Enable	Joint enable control	joint_num	uint8	Joint number, range: 1~6
		state	bool	true-Enable false-Disable

IO_Update	Query the IO input states of the robot arm	type	uint8	0x01-All the IO input states of the controller, 0x02- All IO inputs of the end tool. After the command is issued, the state is obtained through Arm_IO_State/Tool_IO_State.

7.2 Run the demo

Before running, power on RM65-B robot and wait 30 seconds for the robot to complete initialization.

Ensure that the IP of the host computer and the robot arm are in the same local area network, and then can be executed the following command on the terminal.

```
ping 192.168.1.18
```

If you can ping through, you can continue the following operations.

Modify rm_65_moveit_config/launch/rm_65_moveit_controller_manager.launch.xml file to configure and load controllers.yaml file to the parameter server.

```
<launch>
<arg name="execution_type" default="FollowJointTrajectory" />

<!-- loads moveit_controller_manager on the parameter server which is taken as argument
if no argument is passed, moveit_simple_controller_manager will be set -->
<arg                                                              name="moveit_controller_manager"
default="moveit_simple_controller_manager/MoveItSimpleControllerManager" />
<param name="moveit_controller_manager" value="$(arg moveit_controller_manager)"/>

<!-- load controller_list -->
<arg name="use_controller_manager" default="true" />
<param name="use_controller_manager" value="$(arg use_controller_manager)" />

<!-- loads ros_controllers to the param server -->
<!-- <rosparam file="$(find rm_65_moveit_config)/config/ros_controllers.yaml"/> -->
<rosparam file="$(find rm_65_moveit_config)/config/controllers.yaml"/>
<!-- <rosparam file="$(find rm_65_moveit_config)/config/controllers_gazebo.yaml"/> -->
</launch>
```

Open the terminal and execute the following command to run the rm_control node first.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_control rm_control.launch
```

Then open a new terminal and execute the following commands to start the rm_driver node and load MoveIt! rviz.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_bringup rm_robot.launch
```

After running successfully, you can see in rviz that the robot model is consistent with the state of the real robot arm, as shown in Fig.7-1 and Fig. 7-2.

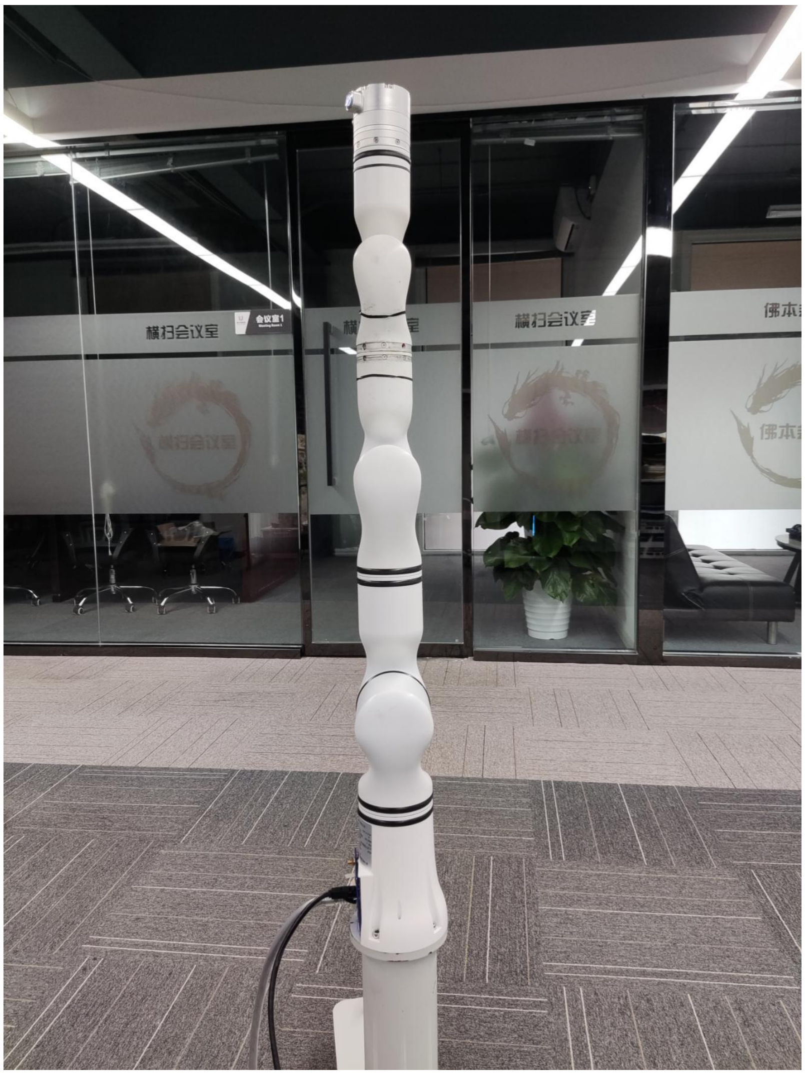
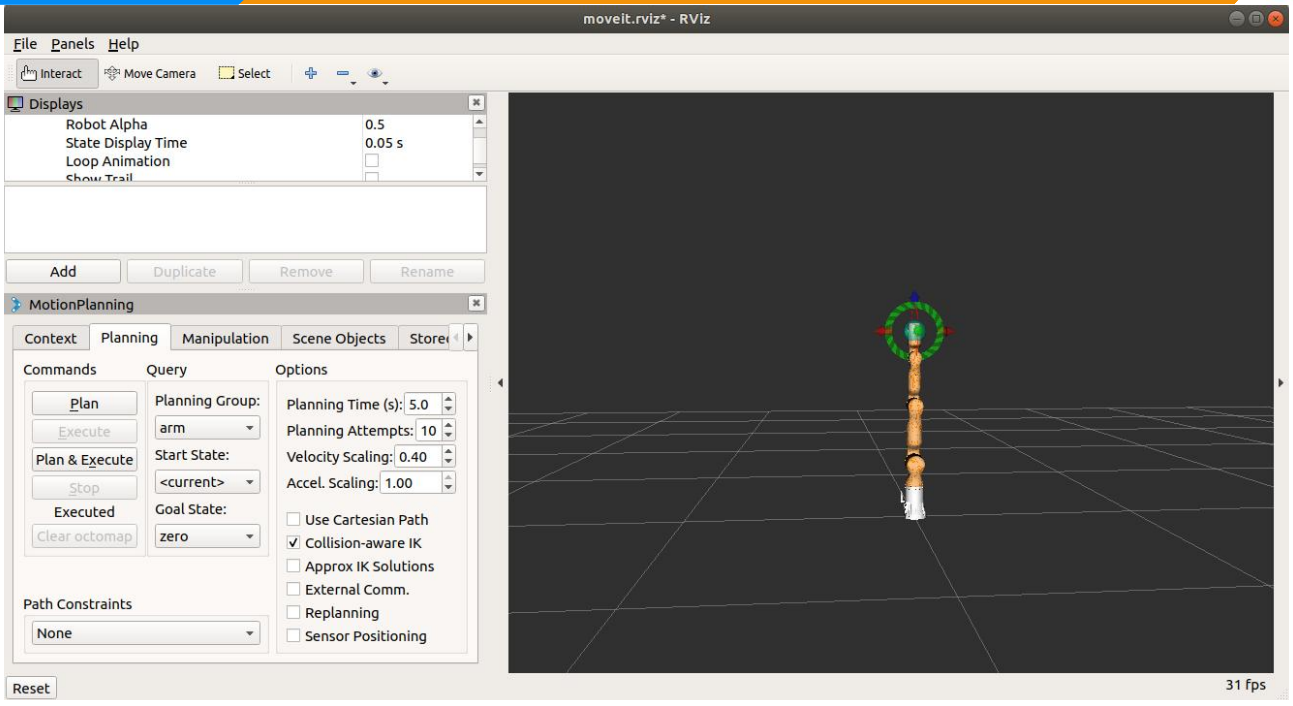


Fig. 7-1 The real robot arm demo.



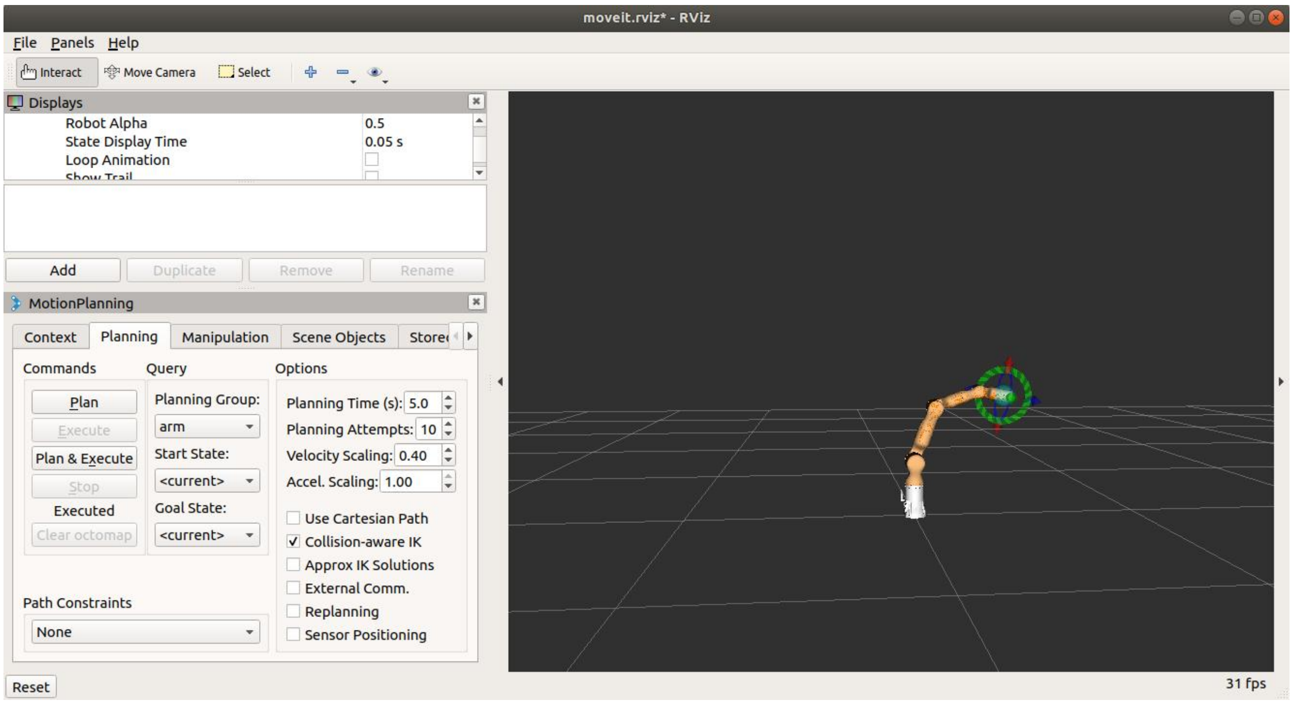
Robot Alpha 0.5
State Display Time 0.05 s
Loop Animation
Show Trail

Planning Group: arm
Start State: <current>
Goal State: zero
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.40
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 7-2 The state of the robot arm model in rviz consistent with the real robot arm.

Drag planning in rviz and then click the "Plan & Execute" button, you can see that the real robot arm will move according to the path planned by MoveIt! in rviz, as shown in Fig.7-3 and Fig. 7-4.



Robot Alpha 0.5
State Display Time 0.05 s
Loop Animation
Show Trail

Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.40
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 7-3 The planned path of the robot arm model running to the specified position in rviz.

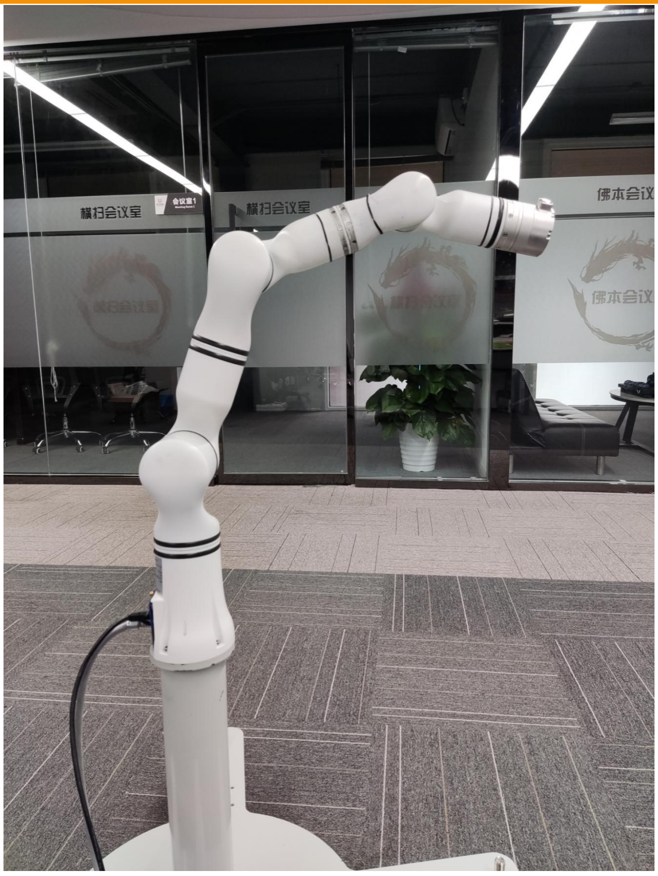


Fig. 7-4 The real robot arm moving to the same position according to the path planned by MoveIt! in rviz.

8. MoveIt! programming demo---scene planning

8.1 Overview

In practical applications, functionalities provided by MoveIt! GUI are limited, and many implementations need to be completed in source code. Move_group of MoveIt! also provides a bunch of C++ and Python programming APIs, which help complete more motion control related functionalities.

8.2 Functionality

1）Add and remove objects in the scene

2）Attach and remove objects for the robot

The program source code ： rm_65_demo/src/planning_scene_ros_api_demo.cpp. See code comments for specific implementation.

8.3 Run the demo

First open the terminal and execute the following command to run MoveIt! demo.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_65_moveit_config demo.launch
```

The opened rviz is shown in Fig. 8-1.

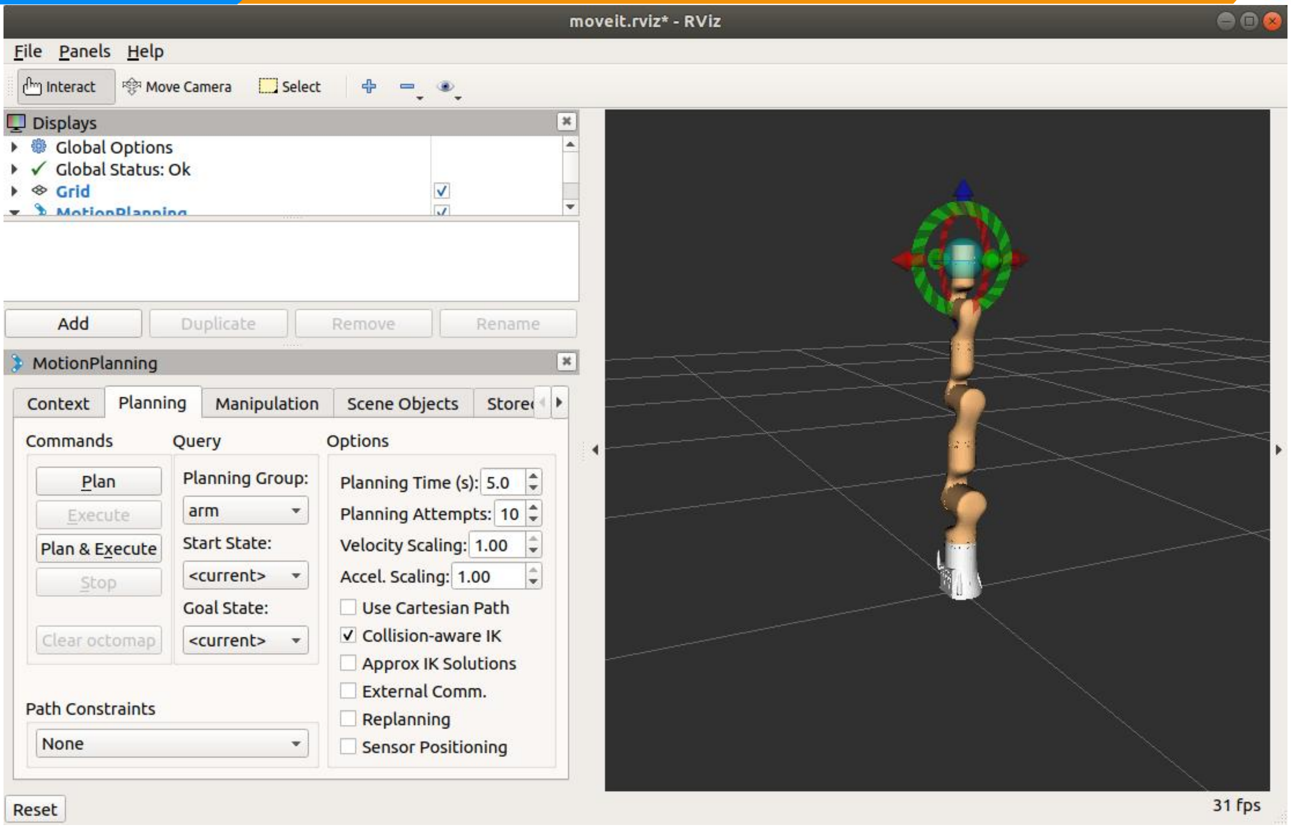


Fig. 8-1 MoveIt! demo interface.

Because this demo uses the MoveItVisualTools plug-in to run, you need to add the RvizVisualToolsGui plug-in to rviz. The addition operation is shown in Fig. 8-2~Fig. 8-4.

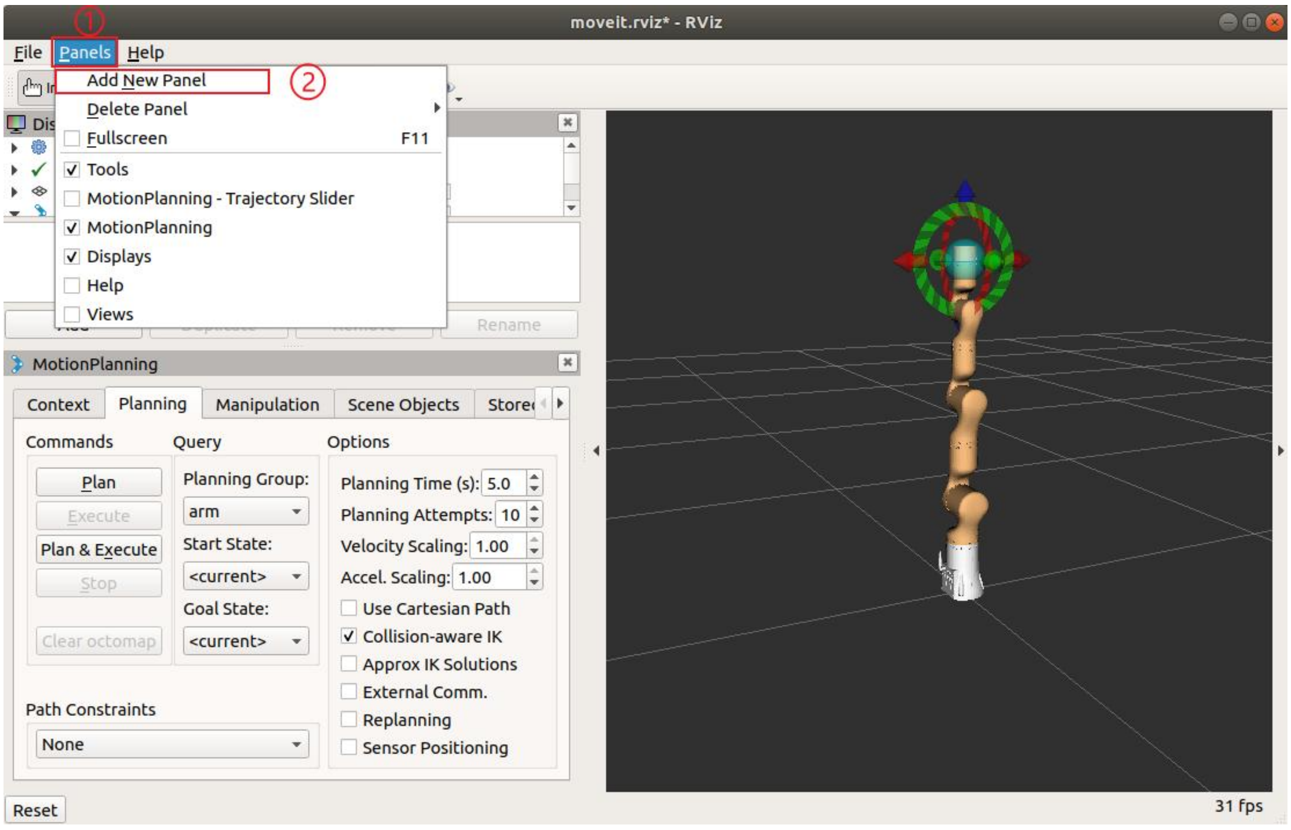


Fig. 8-2 Adding New Panel in rviz.

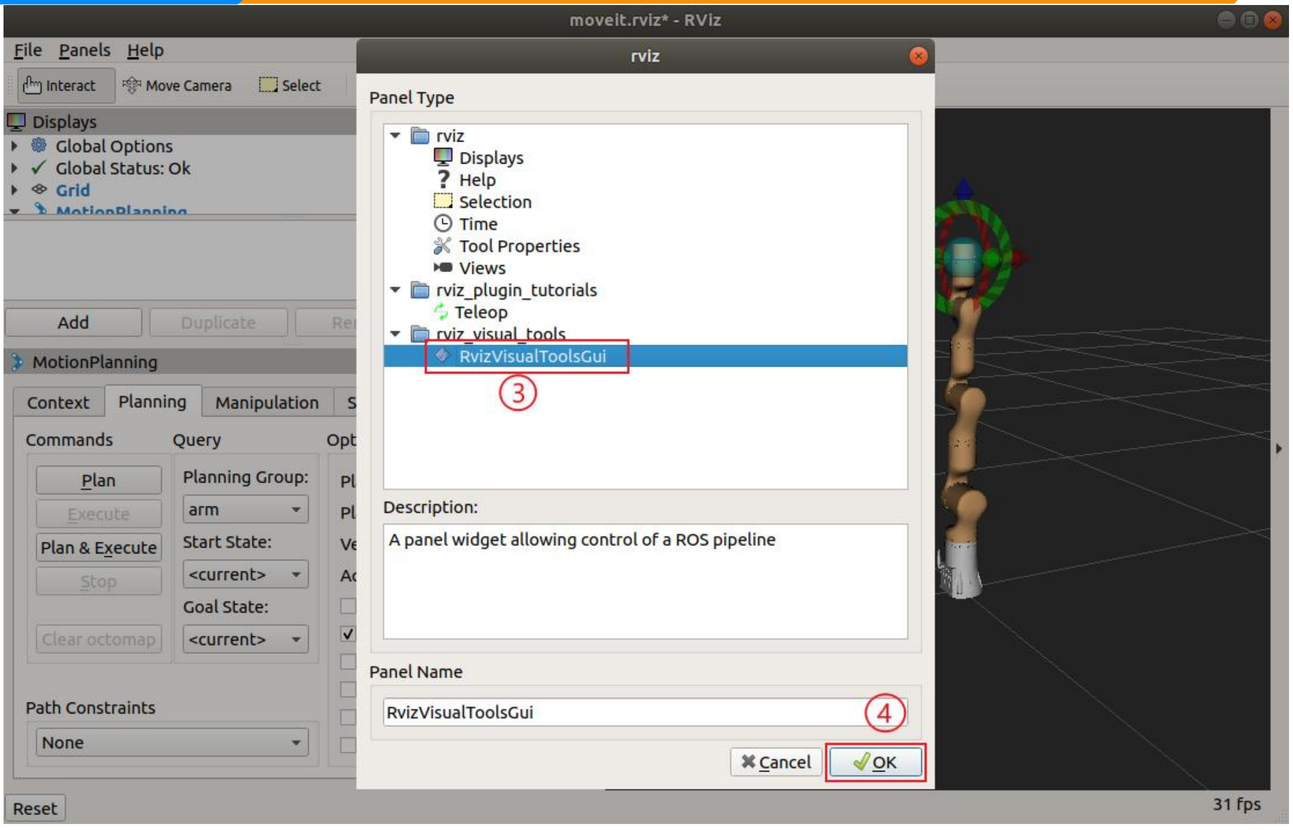


Fig. 8-3 Adding RvizVisualToolsGui in rviz.

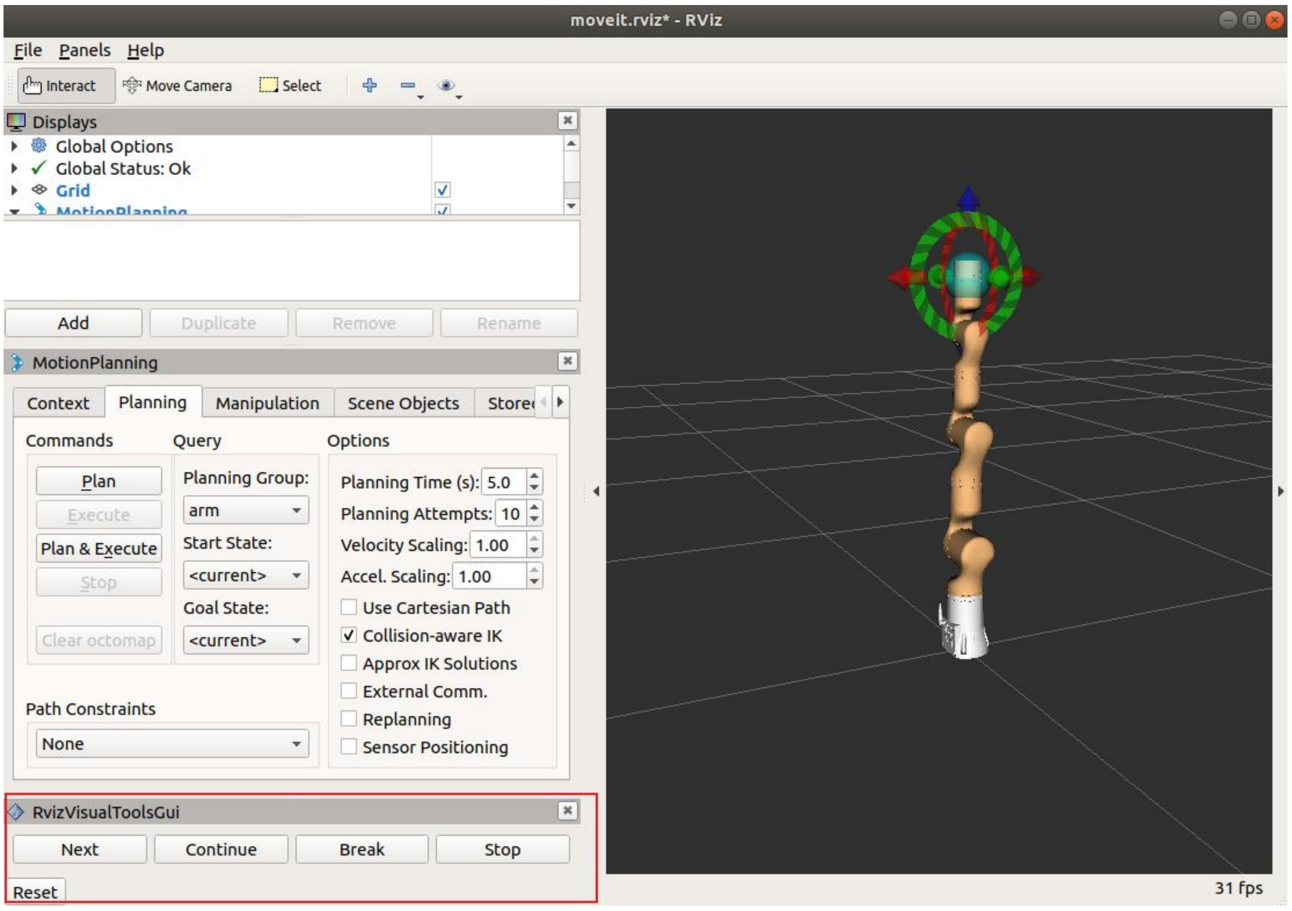
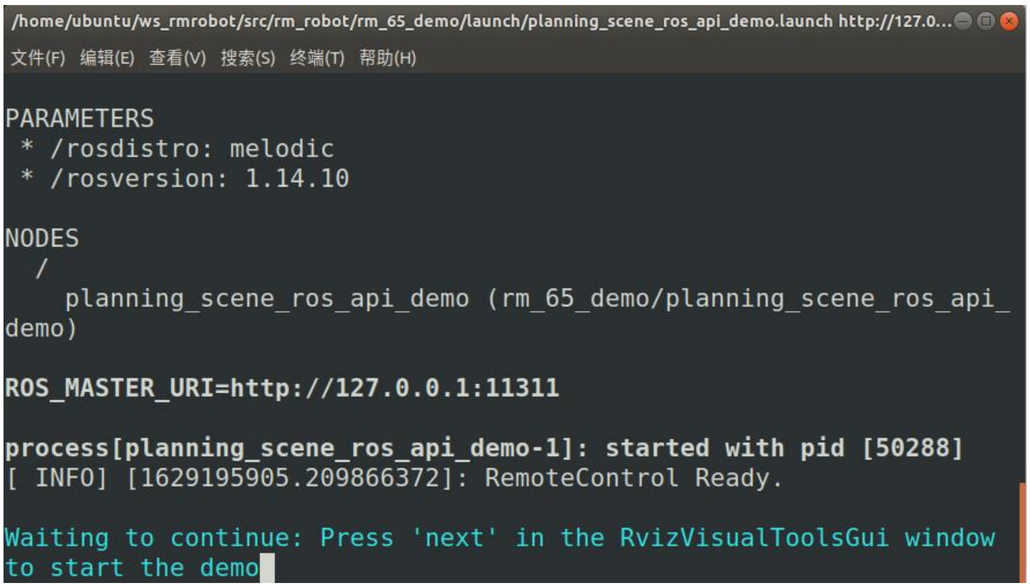


Fig. 8-4 Adding RvizVisualToolsGui in rviz

Open a new terminal and execute the following command to start the scene planning node.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_65_demo planning_scene_ros_api_demo.launch
```

After the scene planning node is started, it is prompted to click the Next button in the RvizVisualToolsGui panel in rviz to start running the program, as shown in Fig.8-5.



```
/home/ubuntu/ws_rmrobot/src/rm_robot/rm_65_demo/launch/planning_scene_ros_api_demo.launch http://127.0...
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES
  /
    planning_scene_ros_api_demo (rm_65_demo/planning_scene_ros_api_demo)

ROS_MASTER_URI=http://127.0.0.1:11311

process[planning_scene_ros_api_demo-1]: started with pid [50288]
[ INFO] [1629195905.209866372]: RemoteControl Ready.

Waiting to continue: Press 'next' in the RvizVisualToolsGui window to start the demo
```

Fig. 8-5 The scene planning node started and waiting for interactive operation.

Click the Next button in the RvizVisualToolsGui panel in rviz, you can see that a green object is added to the end of the robot, indicating that the object is successfully added to the scene, as shown in Fig. 8-6.

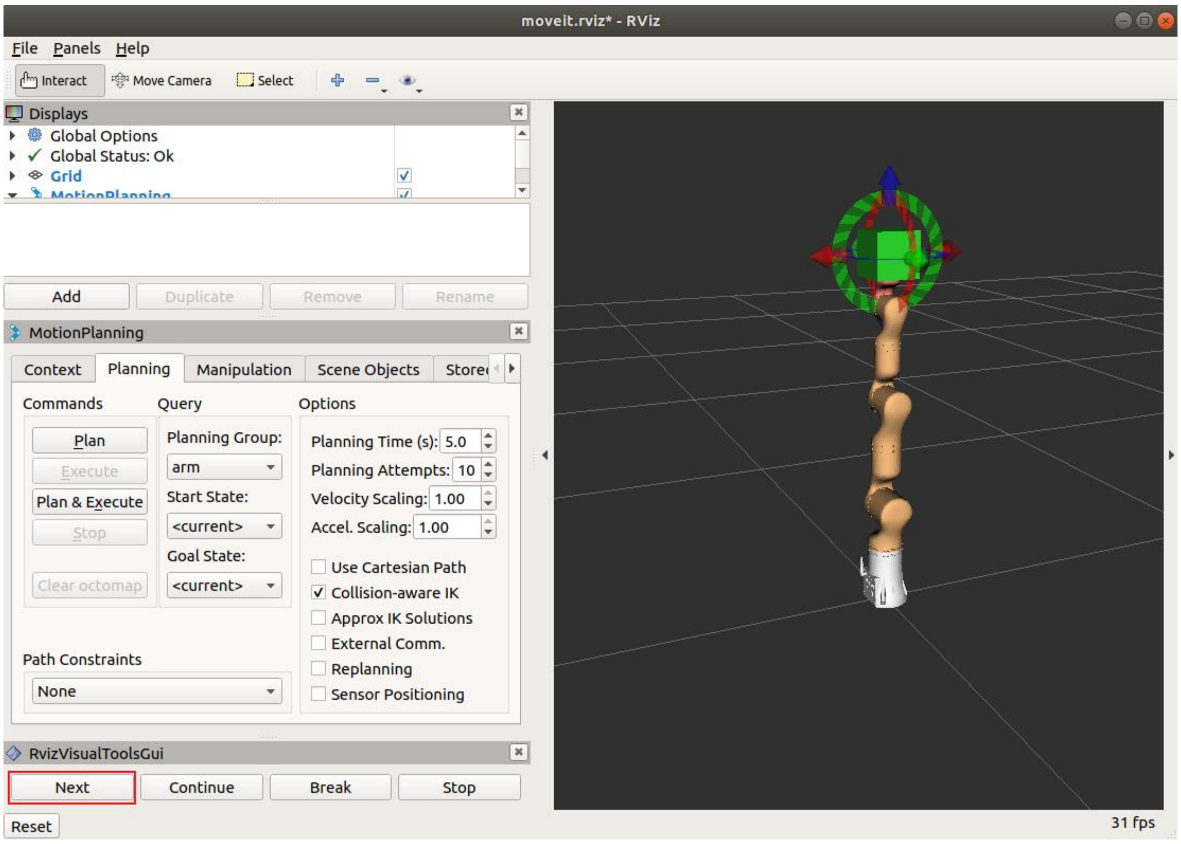
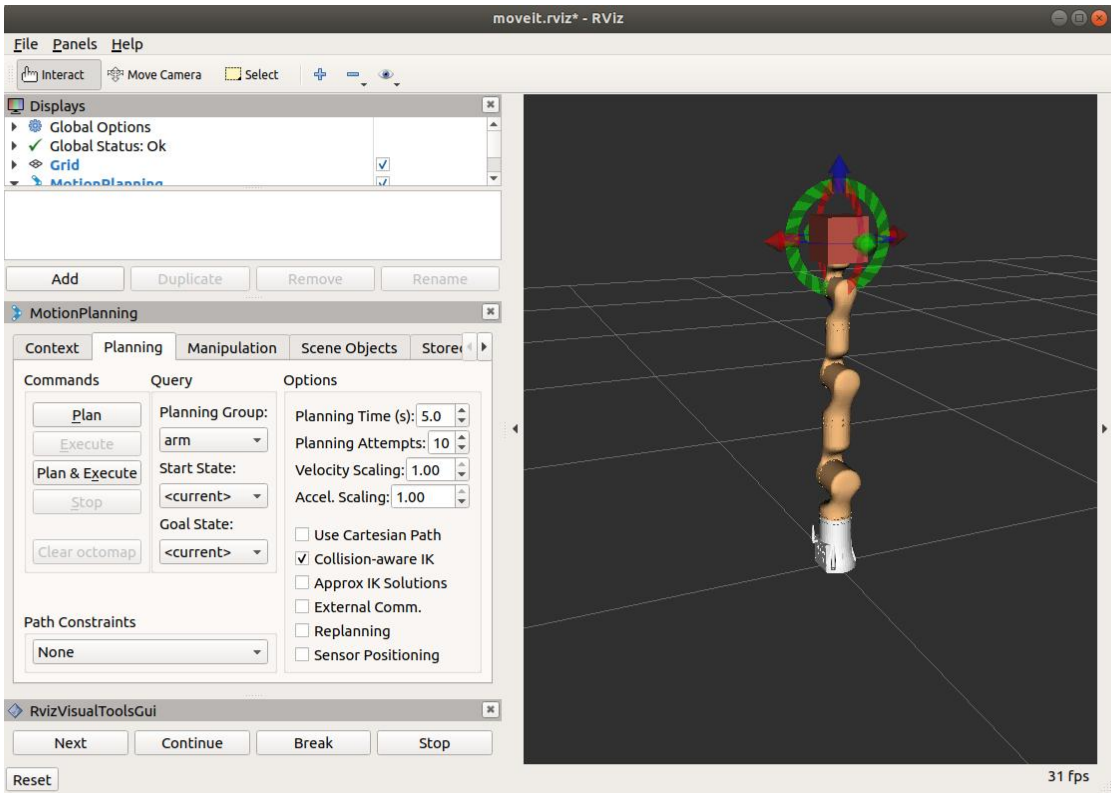


Fig. 8-6 Adding object in scene.

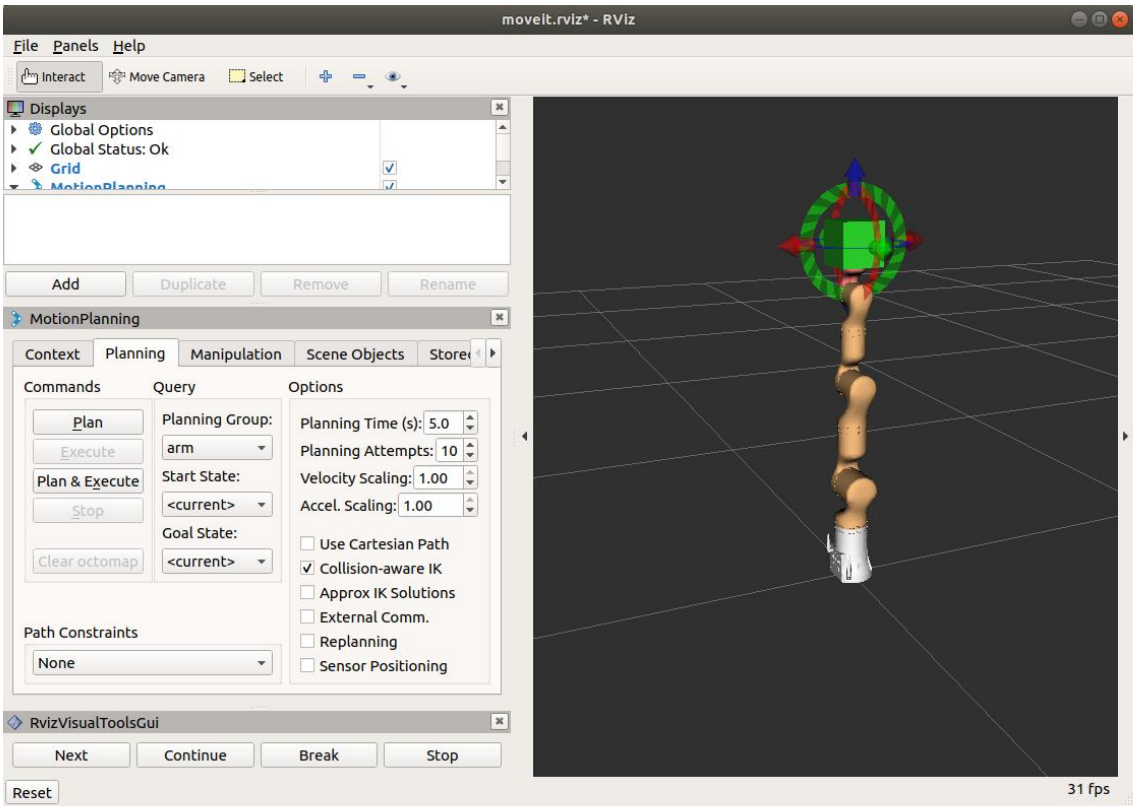
Click the Next button again, and you can see that the object has changed color, which means that the object is attached to the robot, and will be regarded as part of the robot to detect collisions during motion planning, as shown in Fig. 8-7.



Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 8-6 The object attached to the robot.

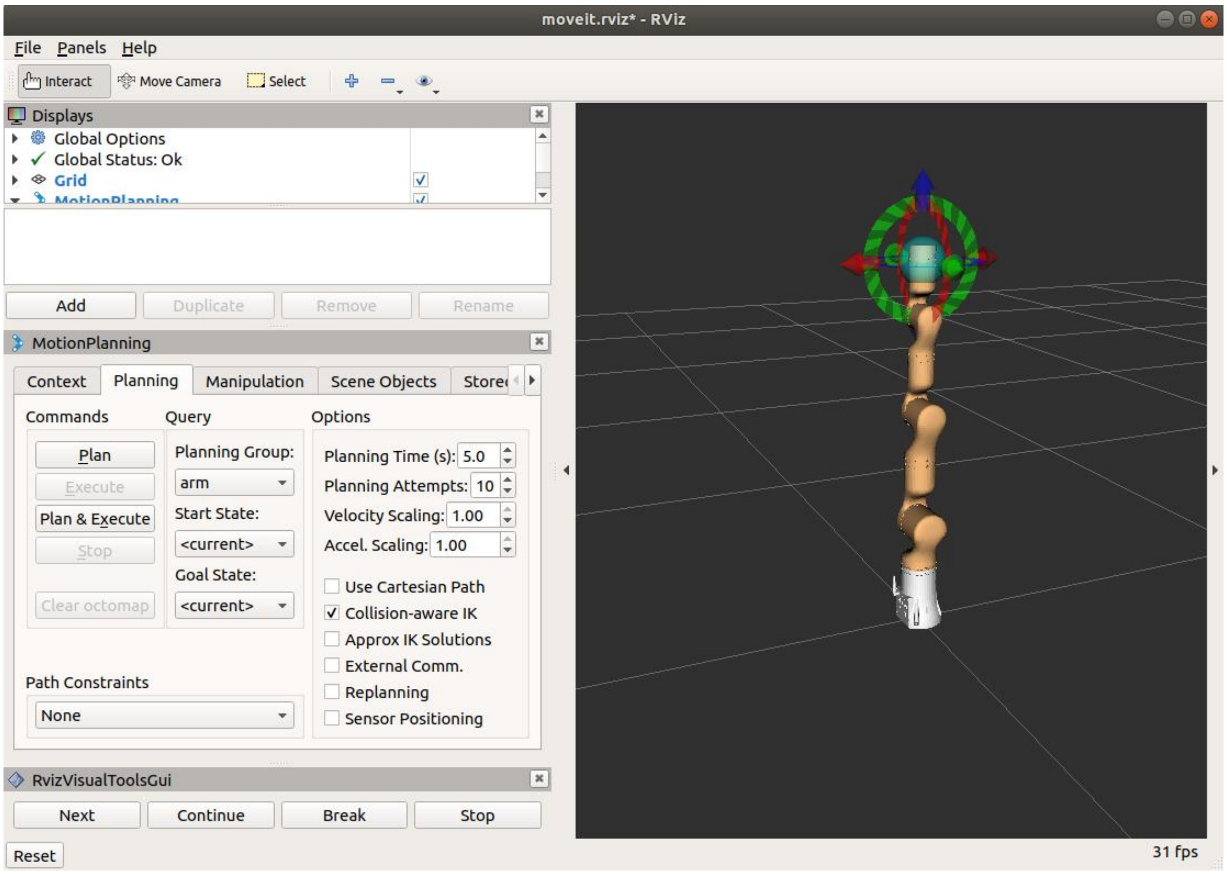
Click the Next button again, you can see that the object turns green again, indicating that the attached object is unattached from the robot, as shown in Fig. 8-7.



Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 8-7 Attached objects unattached.

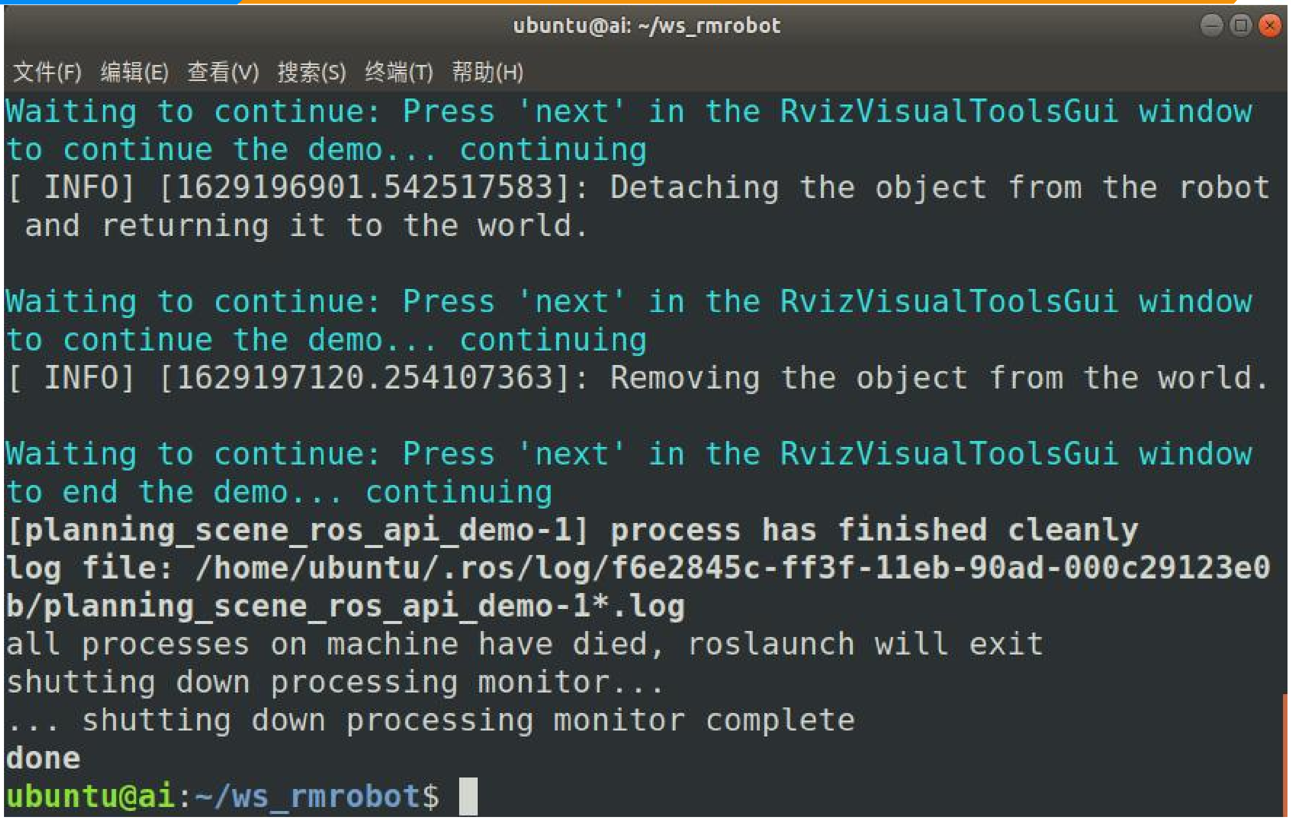
Click the Next button again, you can see that the object has disappeared, indicating that the object has been removed from the scene, as shown in Fig. 8-8.



Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 8-8 The object removed from the scene.

Click the Next button again, the scene planning node exits, as shown in Fig. 8-9.



```
Waiting to continue: Press 'next' in the RvizVisualToolsGui window
to continue the demo... continuing
[ INFO] [1629196901.542517583]: Detaching the object from the robot
 and returning it to the world.

Waiting to continue: Press 'next' in the RvizVisualToolsGui window
to continue the demo... continuing
[ INFO] [1629197120.254107363]: Removing the object from the world.

Waiting to continue: Press 'next' in the RvizVisualToolsGui window
to end the demo... continuing
[planning_scene_ros_api_demo-1] process has finished cleanly
log file: /home/ubuntu/.ros/log/f6e2845c-ff3f-11eb-90ad-000c29123e0
b/planning_scene_ros_api_demo-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
ubuntu@ai:~/ws_rmrobot$
```

Fig. 8-9 Scenario planning node ends and exits.

9. MoveIt! programming demo--- obstacle avoidance planning

9.1 Overview

The source code of this example is Python code, calling the implementation of Python API provided by move_group of MoveIt!. The source code location: rm_65_demo/scripts/moveit_obstacles_demo.py, see code comments for specific implementation.

9.2 Functionality

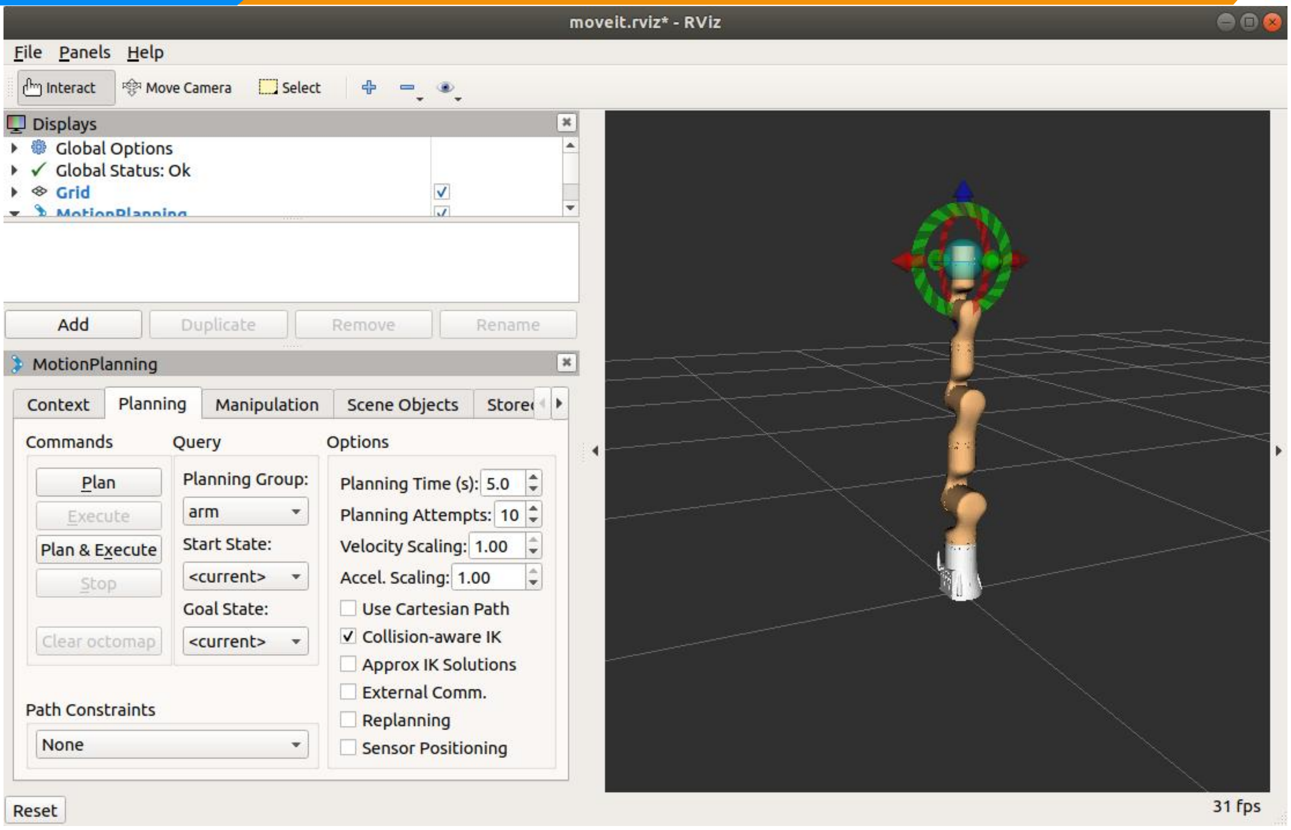
First let the robot move to the predefined "zero" pose, then add a table object to the scene to represent the obstacle, then let the robot automatically avoid the obstacle and move to the "forward" pose, and finally let the robot automatically avoid the obstacle and return to the "zero" position.

9.3 Run the demo

First open the terminal and execute the following command to run the MoveIt! demo.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_65_moveit_config demo.launch
```

The opened rviz is shown in Fig. 9-1.



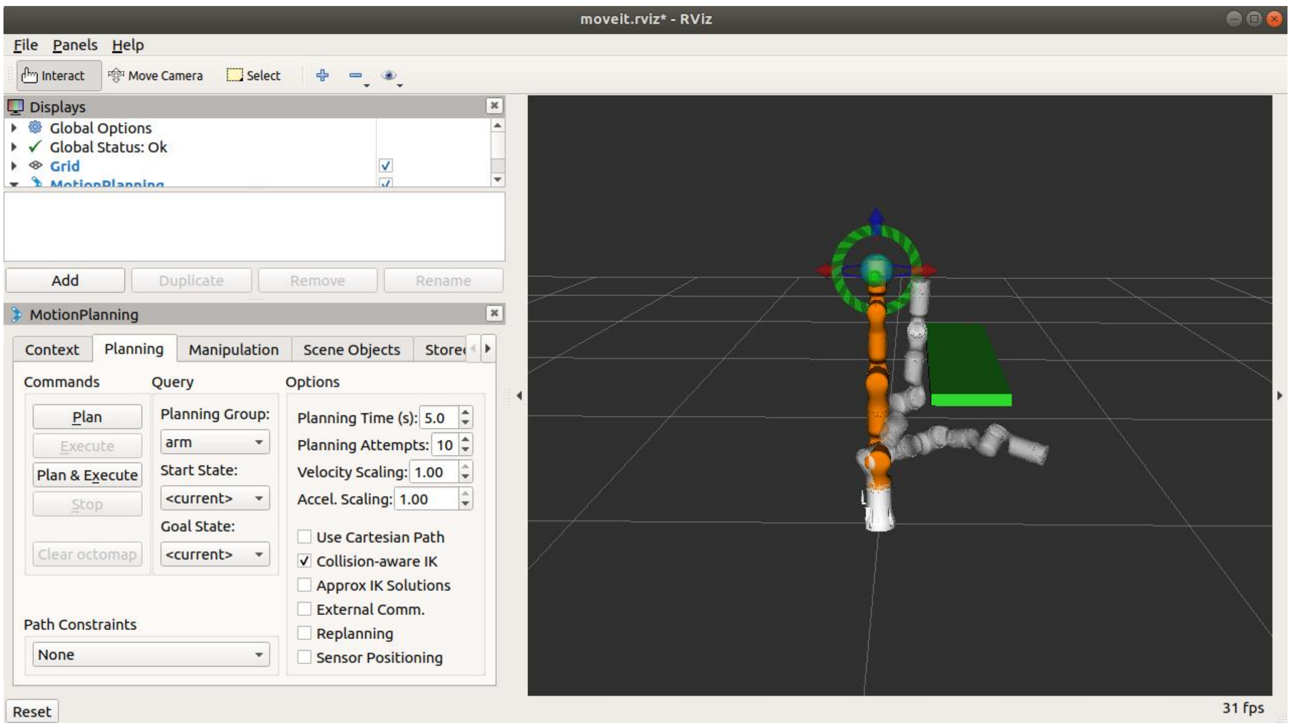
Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 9-1 MoveIt! demo interface.

Open a new terminal and execute the following command to start the obstacle avoidance planning node.

```
rosrun rm_65_demo moveit_obstacles_demo.py
```

After the node runs, you can see in rviz that a table object is added to the scene, and then the robot automatically avoids the table and runs to the forward pose, and finally automatically avoids the table from the forward pose and returns to the zero pose, as shown in Fig. 9- 2.



Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 9-2 Robot autonomous obstacle avoidance demo.

10. MoveIt! programming demo---pick and place

10.1 Overview

The source code of this example is C++ code, calling the C++ API implementation provided by move_group of MoveIt!. The source code location: rm_65_demo/src/pick_place_demo.cpp. See code comments for specific implementation.

10.2 Functionality

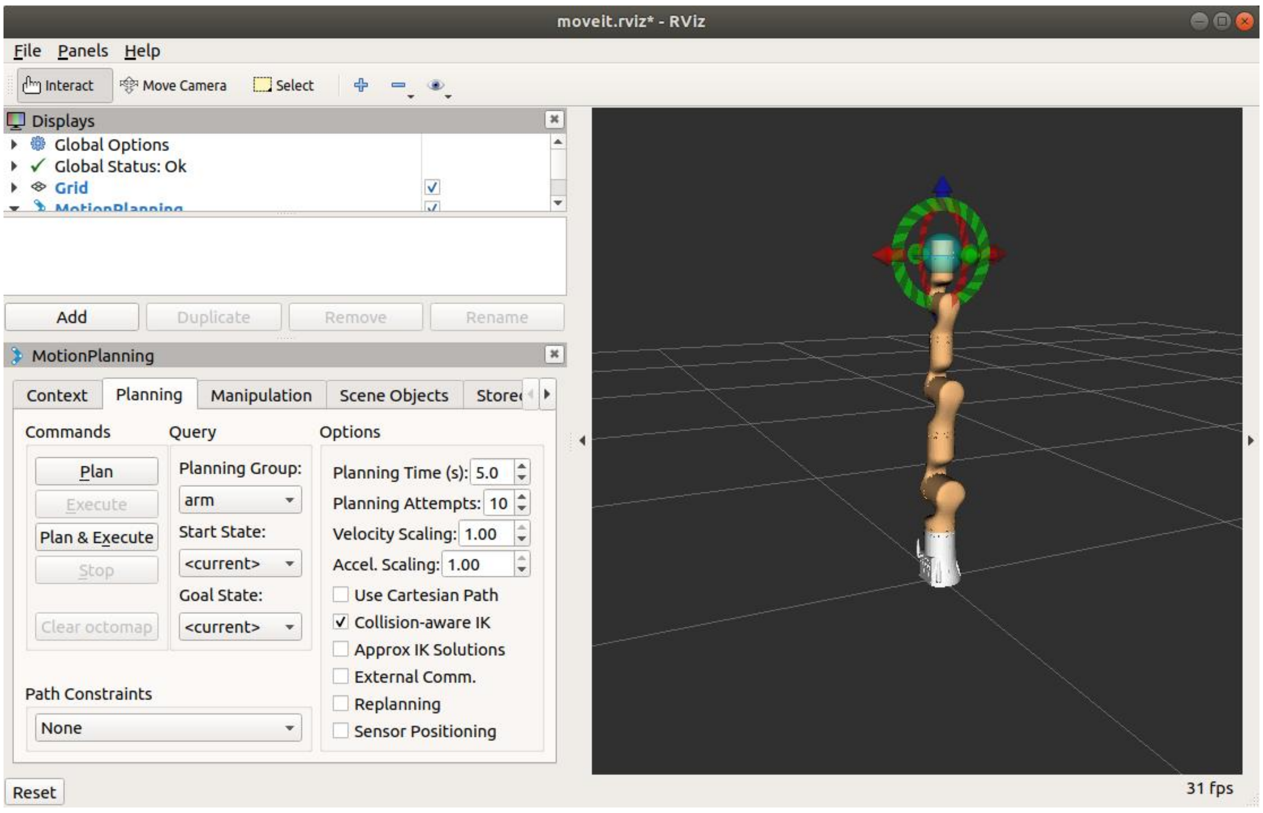
To simulate that robot arm automatically grabs the object and then performs motion planning to place the object in the specified position.

10.3 Run the demo

First open the terminal and execute the following command to run the MoveIt! demo.

```
cd ~/ws_rmrobot/
source devel/setup.bash
roslaunch rm_65_moveit_config demo.launch
```

The opened rviz is shown in Fig. 10-1.



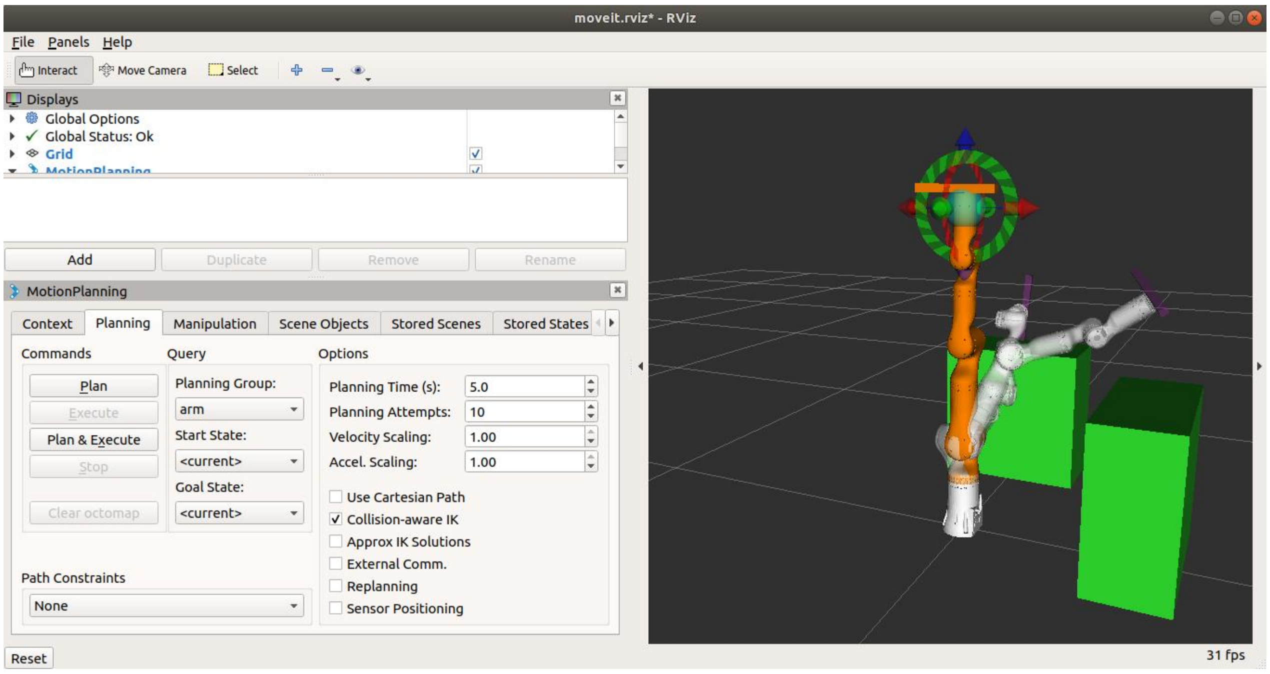
Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Fig. 10-1 MoveIt! demo interface.

Open a new terminal and execute the following command to start the pick_place_demo node.

```
rosrun rm_65_demo pick_place_demo
```

After the node is running, you can see in rviz that three objects have been added to the scene, representing two tables and a grasped target, and then the robot moves to the position of the target and attaches the target to the robot to simulate grasping the object. And then it performs motion planning, places the target object on another table and unattaches it, and finally the robot returns to the zero pose, as shown in Fig. 10-2.



Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00

- [ ] Use Cartesian Path
- [✓] Collision-aware IK
- [ ] Approx IK Solutions
- [ ] External Comm.
- [ ] Replanning
- [ ] Sensor Positioning

Fig. 10-2 Pick and place running demo.