8. MoveIt trajectory planning

This lesson takes the MoveIT simulation as an example. If you need to set the synchronization between the real machine and the simulation, please refer to the lesson [02, MoveIt Precautions and Controlling the Real Machine]. ! ! ! be careful! ! !

The effect demonstration is a virtual machine, and other masters are running (related to the performance of the master, depending on the actual situation).

8.1. Start

Start the MoveIT

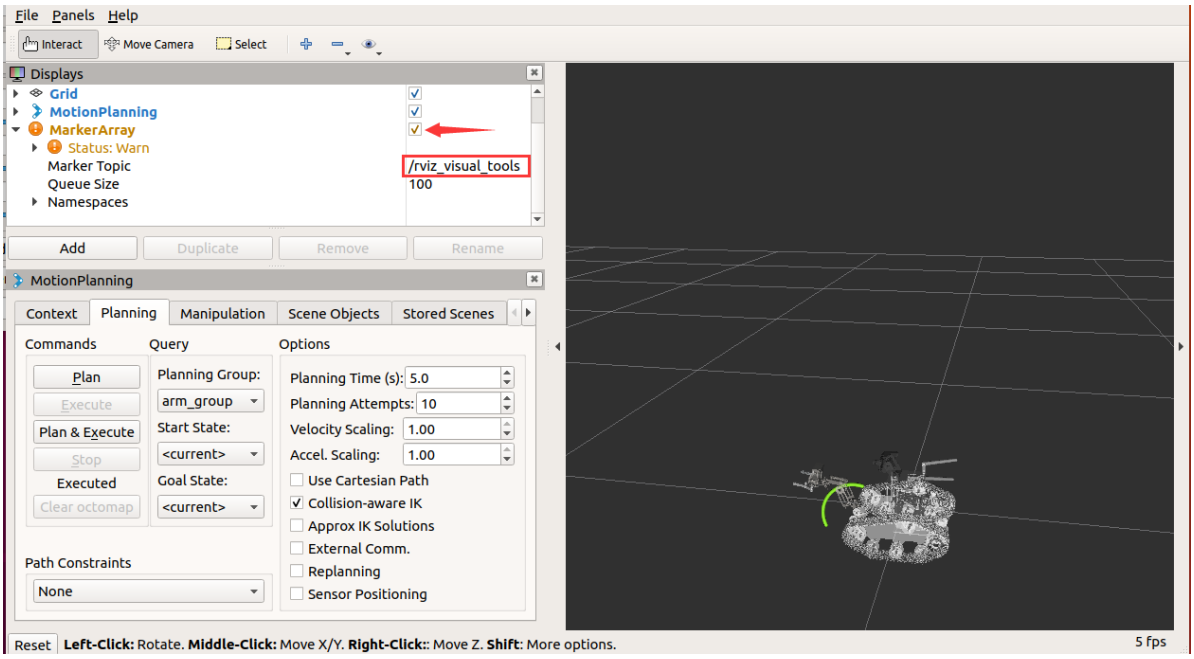
```
roslaunch transbot_se_moveit_config demo.launch
```

Start trajectory planning node

```
rosrun transbot_se_moveit_config 06_multi_track_motion
```

Effect drawing

To view the track, you need to add the [MarkerArray] plugin and select the [/rviz_visual_tools] topic.



Given three reachable target points of the robotic arm, MoveIT will plan three feasible trajectories according to the target points, and then combine the three trajectories into a continuous trajectory.

8.2. Source code analysis

Set three reachable target points (any target point can be reached, it must be reachable)

```
    vector<vector<double>> poses{
            {0.15,  1.62},
            {0,0},
            {2.22, 3.26}
    };
    for (int i = 0; i < poses.size(); ++i) {
        multi_trajectory(transbot, poses.at(i), trajectory);
    }
```

Plan each trajectory

```
void multi_trajectory(
        moveit::planning_interface::MoveGroupInterface &transbot,
        const vector<double> &pose,
        moveit_msgs::RobotTrajectory &trajectory) {
    moveit::planning_interface::MoveGroupInterface::Plan plan;
    const robot_state::JointModelGroup *joint_model_group;
    //Get the robot's starting position
    moveit::core::RobotStatePtr start_state(transbot.getCurrentState());
    joint_model_group = start_state->getJointModelGroup(transbot.getName());
    transbot.setJointValueTarget(pose);
    transbot.plan(plan);
    start_state->setJointGroupPositions(joint_model_group, pose);
    transbot.setStartState(*start_state);
    trajectory.joint_trajectory.joint_names =
plan.trajectory_.joint_trajectory.joint_names;
    for (size_t j = 0; j < plan.trajectory_.joint_trajectory.points.size(); j++)
{

 trajectory.joint_trajectory.points.push_back(plan.trajectory_.joint_trajectory.
points[j]);
    }
}
```

Trajectory merge

```
   moveit::planning_interface::MoveGroupInterface::Plan joinedPlan;
    robot_trajectory::RobotTrajectory rt(transbot.getCurrentState()-
>getRobotModel(), "arm_group");
    rt.setRobotTrajectoryMsg(*transbot.getCurrentState(), trajectory);
    trajectory_processing::IterativeParabolicTimeParameterization iptp;
    iptp.computeTimeStamps(rt, 1, 1);
    rt.getRobotTrajectoryMsg(trajectory);
    joinedPlan.trajectory_ = trajectory;
```

Track display

```
//Show trace
     tool.publishTrajectoryLine(joinedPlan.trajectory_,
transbot.getCurrentState()->getJointModelGroup("arm_group"));
    tool.trigger();
```

Perform trajectory planning

```
if (!transbot.execute(joinedPlan)) {
        ROS_ERROR("Failed to execute plan");
        return false;
    }
```