Trajectory Planning

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 slow and slow to load. It takes 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 is started, in rviz, the robot arm will plan a path to run and display the motion trajectory.

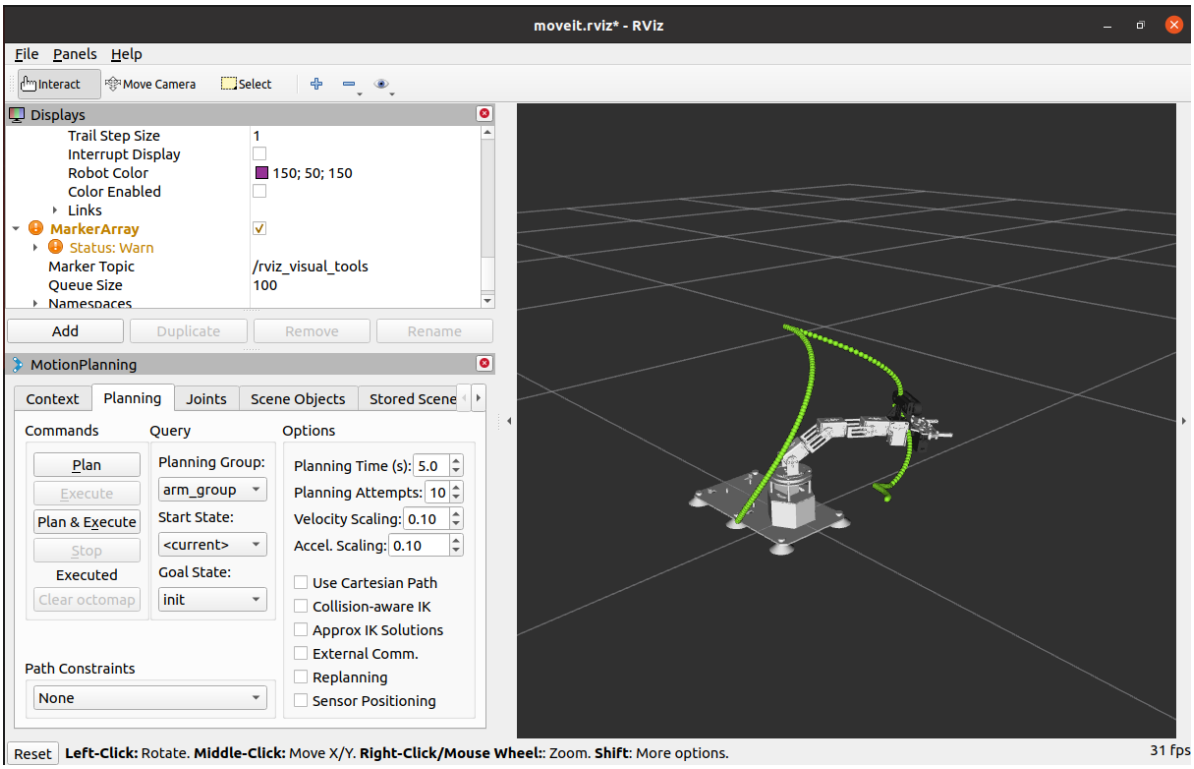
2. Start

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 trajectory planning program
rosrun arm_moveit_demo 06_multi_track_motion
```



Given three reachable target points of the robot arm, MoveIT will plan three feasible trajectories based on the target points, and then merge the three trajectories into a continuous trajectory.

3. Core code analysis

Code path:
/home/jetson/dofbot_pro_ws/src/arm_moveit_demo/src/06_multi_track_motion.cpp

```
#include <ros/ros.h>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit/robot_trajectory/robot_trajectory.h>
#include <moveit/trajectory_processing/iterative_time_parameterization.h>
#include <moveit_msgs/OrientationConstraint.h>
#include <moveit_visual_tools/moveit_visual_tools.h>

using namespace std;

// Plan each trajectory
void multi_trajectory(
        moveit::planning_interface::MoveGroupInterface &dofbot_pro,
        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(dofbot_pro.getCurrentState());
    joint_model_group = start_state->getJointModelGroup(dofbot_pro.getName());
    dofbot_pro.setJointValueTarget(pose);
    dofbot_pro.plan(plan);
    start_state->setJointGroupPositions(joint_model_group, pose);
    dofbot_pro.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]);
    }
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "moveit_revise_trajectory_demo");
    ros::NodeHandle node_handle;
    ros::AsyncSpinner spinner(1);
    spinner.start();
    moveit_msgs::RobotTrajectory trajectory;
    moveit::planning_interface::MoveGroupInterface dofbot_pro("arm_group");
    moveit_visual_tools::MoveItVisualTools tool(dofbot_pro.getPlanningFrame());
    tool.deleteAllMarkers();
    dofbot_pro.allowReplanning(true);
    // Planning time (in seconds)
    dofbot_pro.setPlanningTime(5);
    dofbot_pro.setNumPlanningAttempts(10);
    // Set the allowable target angle error
    dofbot_pro.setGoalJointTolerance(0.01);
    dofbot_pro.setGoalPositionTolerance(0.01);
```

```
    dofbot_pro.setGoalOrientationTolerance(0.01);
    dofbot_pro.setGoalTolerance(0.01);
    // Set the maximum allowed speed and acceleration
    dofbot_pro.setMaxVelocityScalingFactor(1.0);
    dofbot_pro.setMaxAccelerationScalingFactor(1.0);
    // Control the robot arm to return to the initialization position first
    dofbot_pro.setNamedTarget("down");
    dofbot_pro.move();
    // Set three reachable target points (any number of target points is
acceptable, they must be reachable)
    vector<vector<double>> poses{
            {1.34,  -1.0,  -0.61, 0.2,   0},
            {0,     0,     0,     0,     0},
            {-1.16, -0.97, -0.81, -0.79, 3.14}
    };
    for (int i = 0; i < poses.size(); ++i) {
        multi_trajectory(dofbot_pro, poses.at(i), trajectory);
    }
    // Track Merge
    moveit::planning_interface::MoveGroupInterface::Plan joinedPlan;
    robot_trajectory::RobotTrajectory rt(dofbot_pro.getCurrentState()-
>getRobotModel(), "arm_group");
    rt.setRobotTrajectoryMsg(*dofbot_pro.getCurrentState(), trajectory);
    trajectory_processing::IterativeParabolicTimeParameterization iptp;
    iptp.computeTimeStamps(rt, 1, 1);
    rt.getRobotTrajectoryMsg(trajectory);
    joinedPlan.trajectory_ = trajectory;
    // Show Tracks
    tool.publishTrajectoryLine(joinedPlan.trajectory_,
dofbot_pro.getCurrentState()->getJointModelGroup("arm_group"));
    tool.trigger();
    // Execute trajectory planning
    if (!dofbot_pro.execute(joinedPlan)) {
        ROS_ERROR("Failed to execute plan");
        return false;
    }
    sleep(1);
    ROS_INFO("Finished");
    ros::shutdown();
    return 0;
}
```