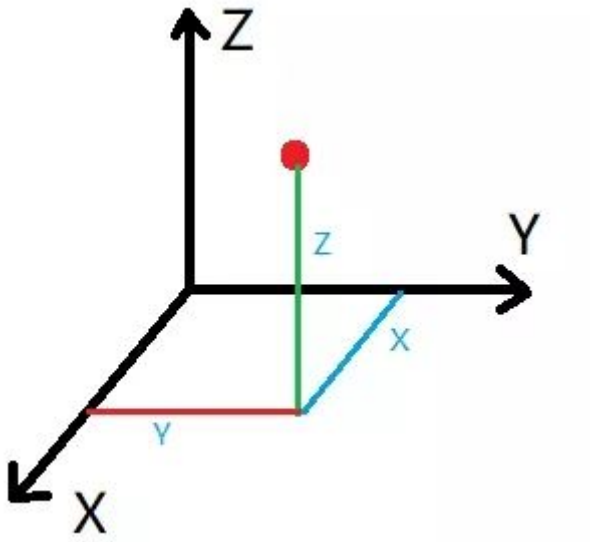
5. MoveIt Cartesian Path

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).

5.1. Introduction

The Cartesian coordinate system is the collective name for the Cartesian coordinate system and the oblique coordinate system. A Cartesian path is actually a line connecting any two points in space



5.2. Start

Start the MoveIT

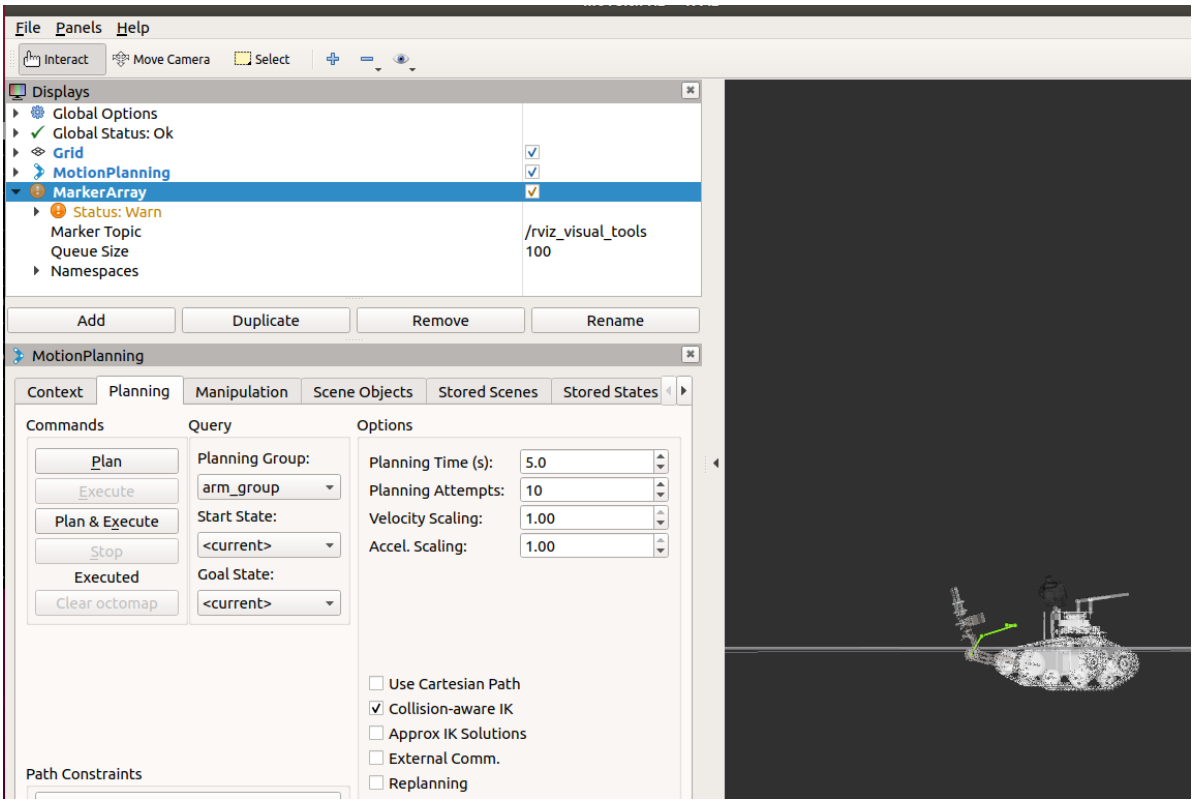
```
roslaunch transbot_se_moveit_config demo.launch
```

Start the Cartesian path node

```
rosrun transbot_se_moveit_config 04_cartesian    # C++
rosrun transbot_se_moveit_config 04_cartesian.py # python
```

- C++ code example

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



- Python code example

The python code does not have a similar trajectory to C++, and there is no special effect after running.

5.3. Source code

5.3.1. py file

Set specific location

```
    rospy.loginfo("Set Init Pose")
    joints = [0.49, 0.93]
    transbot.set_joint_value_target(joints)
    transbot.execute(transbot.plan())
```

Add waypoint

```
    # Initialize waypoint list
    waypoints = []
    # If True, add the initial pose to waypoint list
    waypoints.append(start_pose)
    wpose = deepcopy(start_pose)
    wpose.position.x =0.00055
    wpose.position.y =0.1139
    wpose.position.z =0.049213
    waypoints.append(deepcopy(wpose))
```

```
    wpose.position.x =0.00055
    wpose.position.y =0.11303
    wpose.position.z =0.049142
    waypoints.append(deepcopy(wpose))

    wpose.position.x =0.00055
    wpose.position.y =0.10873
    wpose.position.z =0.048655
    waypoints.append(deepcopy(wpose))

    wpose.position.x =0.00055
    wpose.position.y =0.10616
    wpose.position.z =0.048255
    waypoints.append(deepcopy(wpose))


    wpose.position.x =0.00055
    wpose.position.y =0.10446
    wpose.position.z =0.047944
    waypoints.append(deepcopy(wpose))
```

Waypoint Planning

```
        '''
        waypoints:waypoints:waypoint list
        eef_step:  terminal step value, calculate the inverse solution every 0.1m
to determine whether it is reachable
        jump_threshold: jump threshold, set to 0 means jumping is not allowed
        plan: path, fraction:  path planning coverage
        '''
        (plan, fraction) = transbot.compute_cartesian_path(waypoints, 0.1, 0.0,
True)
```

5.3.1.C++ file

Set specific location

```
    ROS_INFO("Set Init Pose.");
    //Set specific location
    vector<double> pose{0.28, 2.14};
    transbot.setJointValueTarget(pose);
```

Add waypoint

```
    //Initialize waypoint vector
    std::vector<geometry_msgs::Pose> waypoints;
    //Add the initial pose to the waypoint list
   waypoints.push_back(start_pose);

start_pose.position.x = 0.00055;
start_pose.position.y = 0.10701;
start_pose.position.z = 0.048397;
waypoints.push_back(start_pose);
```

```
start_pose.position.x = 0.00055;
start_pose.position.y = 0.17728;
start_pose.position.z = 0.026649;
waypoints.push_back(start_pose);

start_pose.position.x = 0.00055;
start_pose.position.y = 0.19944;
start_pose.position.z = -0.009218;
waypoints.push_back(start_pose);
```

Waypoint planning

```
fraction = transbot.computeCartesianPath(waypoints, eef_step, jump_threshold,
trajectory);
```