3、Robotic arm avoiding

Website：http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index.html

HD camera MoveIt：~/software/transbot_library/src/transbot_config_camera

Astra_MoveIt：~/software/transbot_library/src/transbot_config_astra

Robotic arm control function package：~/software/transbot_library/src/transbot_description

Before usingg this function, we need to close the APP remote control process and all the functions that have been turned on. MoveIT recommends running in a virtual machine.

(The computer must have a discrete GPU!!!)

3.1、Start

This lesson is mainly to learn the random movement MoveIT simulation.

Taking the HD camera configuration as an example, the Astra configuration is similar.

Virtual machine side

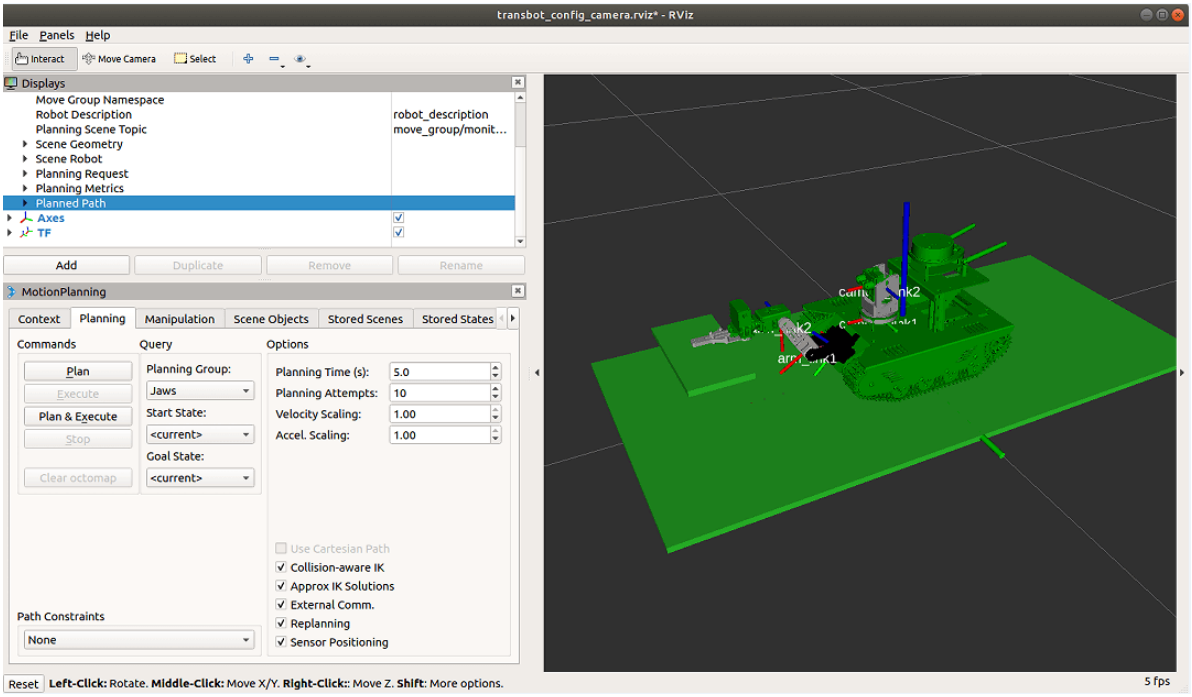
```
roslaunch transbot_config_camera demo.launch        # HD camera
roslaunch transbot_config_astra demo.launch         # astra
rosrun transbot_description 02_attached_object      # C++
rosrun transbot_description 02_attached_object.py   # python
```

If you need to combine the real machine, you need to configure multi-machine communication, and it is very dangerous and easy to damage the robot.
Inexperienced users are not recommended to connect to the real machine and move randomly.

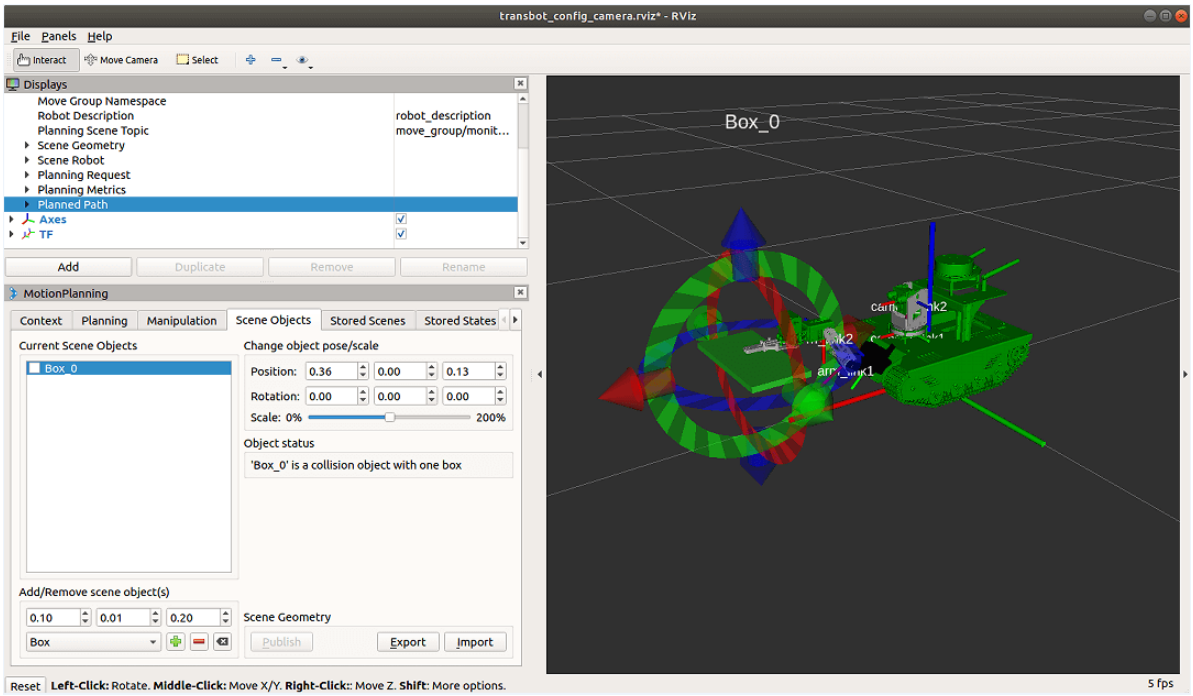
Drive the real machine（jetson nano）

```
rosrun transbot_description 03_machine_move.py
```

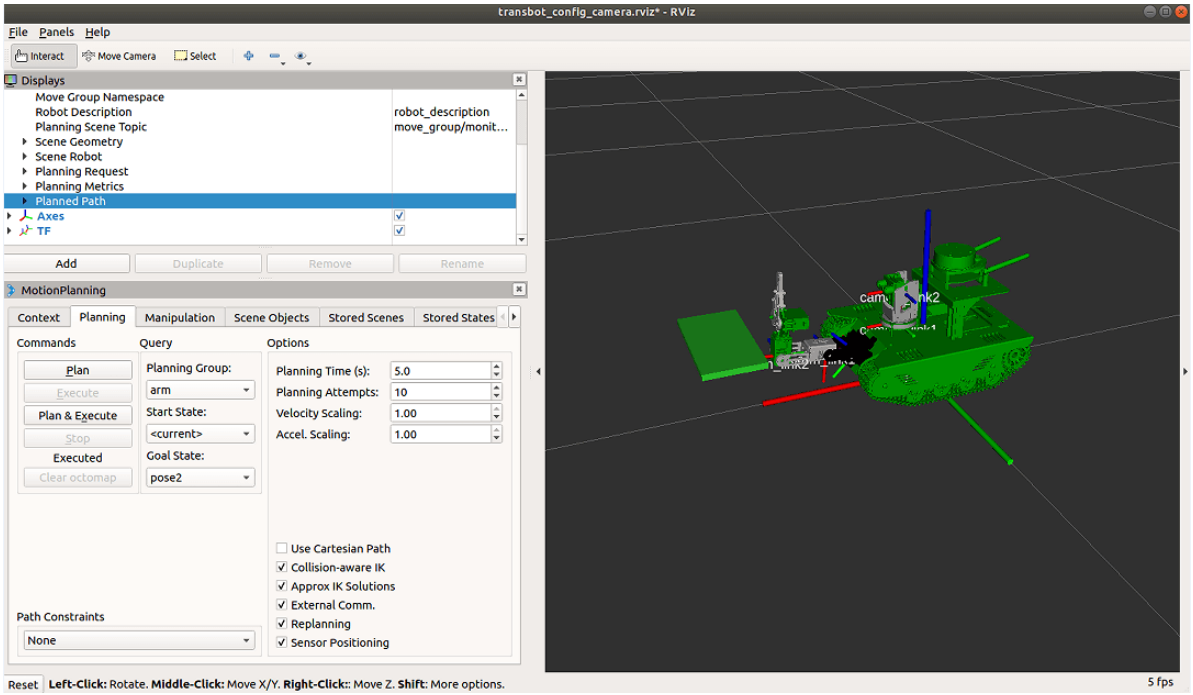
- Load code



- Add to rviz
  - Click【SceneObjects】in 【MotionPlanning】to enter scene settings,
  - Choose【Box】property,
  - Adjust scene object(s)parameter【0.10，0.01，0.2】
  - Click（+）to add obstacles；Click（-）or（×）to remove obstacles
  - Adjust 【Position】property【0.36，0.00，0.13】，change the pose of the cabinet.



Go back to 【Planning】and plan the robotic arm again. It will never collide with the cuboid, nor can it reach the position of the cuboid.



3.2、Source code analysis

py file

- Basic Settings

```
# Initialize move_group API
moveit_commander.roscpp_initialize(sys.argv)
# Initialize the ROS node
rospy.init_node('attached_object_py')
# Initialize the scene object
scene = PlanningSceneInterface()
# Initialize robotic arm
transbot = MoveGroupCommander('arm')
# When motion planning fails, re-planning is allowed
transbot.allow_replanning(True)
transbot.set_planning_time(5)
# Number of planning attempts
transbot.set_num_planning_attempts(10)
# Set the allowable target position error
transbot.set_goal_position_tolerance(0.01)
# Set the allowable target attitude error
transbot.set_goal_orientation_tolerance(0.01)
# Set the allowable target error
transbot.set_goal_tolerance(0.01)
# Set maximum speed
transbot.set_max_velocity_scaling_factor(1.0)
# Set maximum acceleration
transbot.set_max_acceleration_scaling_factor(1.0)
```

- Set "pose1" as the target point

```
# Set "pose1" as the target point
transbot.set_named_target("pose1")
transbot.go()
sleep(0.5)
```

- Add obstacles

```
    # Set the height of the desktop
    table_ground = 0.12
    # Set the 3D size of obstacles [length, width, and height]
    table_size = [0.1, 0.2, 0.01]
    floor_size = [1, 0.5, 0.01]
    # Add the table to the scene
    table_pose = PoseStamped()
    table_pose.header.frame_id = 'base_footprint'
    table_pose.pose.position.x = 0.36
    table_pose.pose.position.y = 0
    table_pose.pose.position.z = table_ground + table_size[2] / 2.0
    table_pose.pose.orientation.w = 1.0
    scene.add_box('table', table_pose, table_size)
    floor_pose = PoseStamped()
    floor_pose.header.frame_id = 'base_footprint'
    scene.add_box('floor', floor_pose, floor_size)
```

C++ file

- Basic Settings

```
    // Initialize the ROS node
    ros::init(argc, argv, "attached_object_cpp");
    // Create node handle
    ros::NodeHandle n;
    // Set thread
    ros::AsyncSpinner spinner(1);
    // Start thread
    spinner.start();
    // Initialize the robotic arm
    moveit::planning_interface::MoveGroupInterface transbot("arm");
    // Allow replanning
    transbot.allowReplanning(true);
    // Planning time (unit: second)
    transbot.setPlanningTime(5);
    //Set the number of planning attempts
    transbot.setNumPlanningAttempts(10);
    //Set the allowable target attitude error (unit: meter)
    transbot.setGoalPositionTolerance(0.01);
    //Set the allowable target position error (unit: radians)
    transbot.setGoalOrientationTolerance(0.01);
    //Set maximum speed
    transbot.setMaxVelocityScalingFactor(1.0);
    //Set maximum acceleration
    transbot.setMaxAccelerationScalingFactor(1.0);
```

- Set "pose1" as the target point

```
    //Set the target point
    transbot.setNamedTarget("pose1");
    //Start moving
    transbot.move();
    sleep(0.1);
```

- Add obstacles

```
    //////////////////////Add obstacles/////////////////////
    vector<string> object_ids;
    scene.removeCollisionObjects(object_ids);
    //Create the detection object container
    vector<moveit_msgs::CollisionObject> objects;
    //Create collision detection object
    moveit_msgs::CollisionObject obj;
    //Set the id of the obstacle
    obj.id = "obj";
    object_ids.push_back(obj.id);
    //The state of the obstacle
    obj.operation = obj.ADD;
    //Set the header information of the obstacle
    obj.header.frame_id = frame;
    shape_msgs::SolidPrimitive primitive;
    shape_msgs::SolidPrimitive floor;
    //Set obstacle type
    primitive.type = primitive.BOX;
    floor.type = primitive.BOX;
    //Set obstacle dimensions
    primitive.dimensions.resize(3);
    floor.dimensions.resize(3);
    //Set the length, width and height of obstacles
    primitive.dimensions[0] = 0.2;
    primitive.dimensions[1] = 0.1;
    primitive.dimensions[2] = 0.01;
    obj.primitives.push_back(primitive);
    floor.dimensions[0] = 1;
    floor.dimensions[1] = 0.5;
    floor.dimensions[2] = 0.01;
    obj.primitives.push_back(floor);
    geometry_msgs::Pose pose;
    geometry_msgs::Pose pose1;
    //Set the location information of obstacles [x,y,z]
    pose.position.x = 0.36;
    pose.position.y = 0;
    pose.position.z = 0.125;
    //Set the posture information of obstacles
    tf::Quaternion quaternion;
    //The unit of R, P, Y is angle
    double Roll = 0.0;
    double Pitch = 0.0;
    double Yaw = 90.0;
    //Convert RPY to quaternion
    quaternion.setRPY(Roll * M_PI / 180, Pitch * M_PI / 180, Yaw * M_PI / 180);
    pose.orientation.x = quaternion.x();
    pose.orientation.y = quaternion.y();
    pose.orientation.z = quaternion.z();
    pose.orientation.w = quaternion.w();
    //Set the pose information of obstacles
    obj.primitive_poses.push_back(pose);
    obj.primitive_poses.push_back(pose1);
    objects.push_back(obj);
    //////////////////////Set the color of obstacles/////////////////////
    //Create a color container for the detection object
```

```
    std::vector<moveit_msgs::ObjectColor> colors;
    //Create a color instance
    moveit_msgs::ObjectColor color;
    //Add the id that needs to set the color
    color.id = "obj";
    //Set RGBA value, range [0~1]
    color.color.r = 0;
    color.color.g = 1.0;
    color.color.b = 0;
    color.color.a = 0.5;
    colors.push_back(color);
    //Add the set information to the scene
    scene.applyCollisionObjects(objects, colors);
```