Driving the real machine

Preface

We have built the MoveIt environment 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 more laggy and slow to load, and it will take 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 tutorial will take running on the Orin motherboard as an example.

1. Functional Description

After the program is started, the robot arm will follow the simulated robot arm in MoveIt. Note that since the real robot arm does not have the obstacle avoidance function, it may encounter obstacles when following the simulated robot arm. Therefore, when driving the simulated robot arm in Rviz, the planned action should not be too large. It is recommended to only test the three preset postures, namely [up], [down] and [init]. Before starting, you need to make sure there are no obstacles around the robot arm to prevent it from hitting obstacles.

2. Start

2.1. Configure virtual machine motherboard multi-machine communication

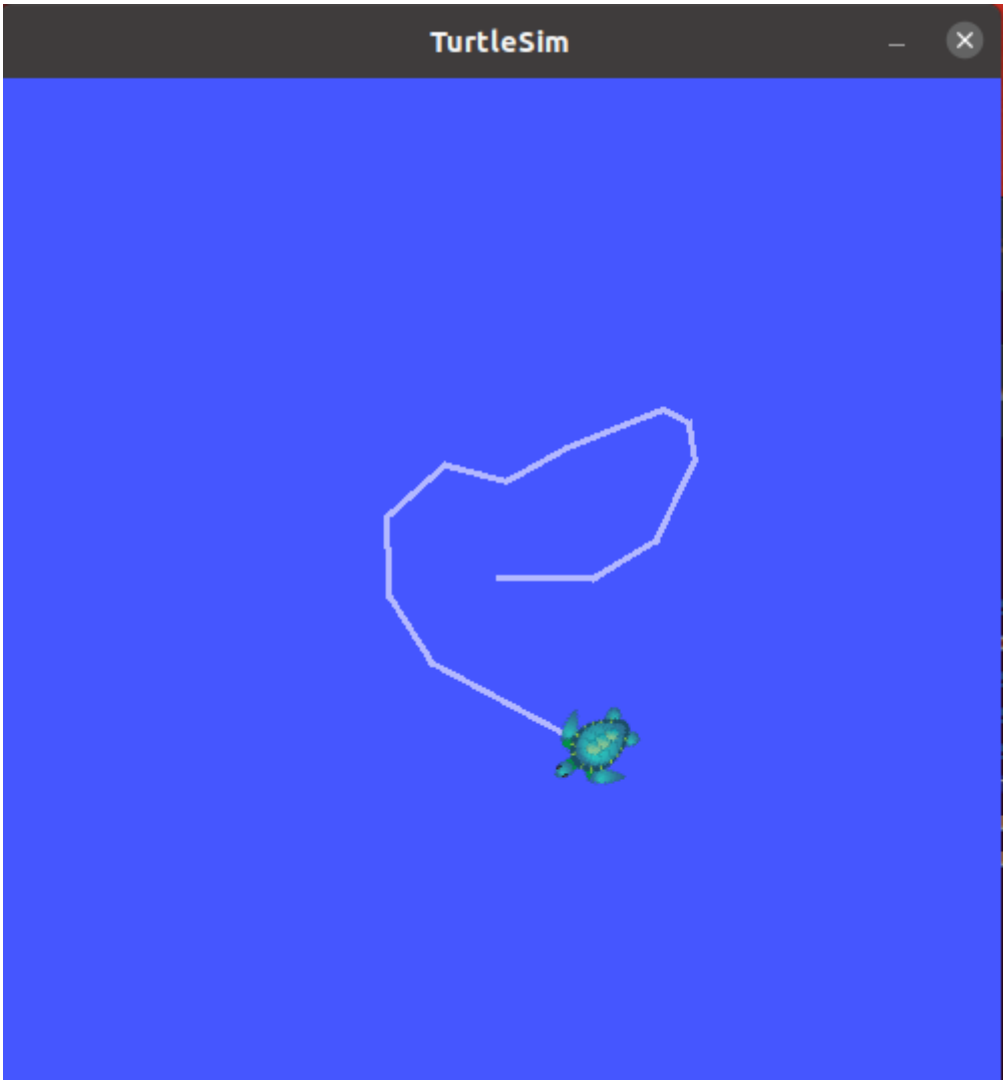
If the Jetson_Nano user starts MoveIt in the virtual machine for testing, you need to set up multi-machine communication. The configuration process is as follows:

- First, you need to ensure that the virtual machine and the Jetson_Nano motherboard are connected to the same network;
- Make sure that the virtual machine is the host and the Jetson_Nano motherboard is the slave. The ROS_IP of the slave needs to be set to the IP of the host. The network IP of the virtual machine can be queried through the ifconfig command. For example, my virtual machine network IP is 192.168.1.108, so I need to set Jetson_Nano's ROS_IP to 192.168.1.108. The setting method is as follows: modify the ~/.bashrc file,

```
sudo vim ~/.bashrc
```

Press [i] or [I] to enter the edit mode, add `export ROS_MASTER_URI=http://192.168.1.108:11311` at the bottom, and change 192.168.1.108 here to the network queried by the virtual machine. After editing, press [Shift] + [:], enter `wq`, save and exit. Then reopen a terminal to refresh the environment variables.

- Verify whether the configuration is successful. Enter `roscore` on the host to start the Master, and enter `rosrun turtlesim turtlesim_node` on the host to start the turtle node. Enter `rosrun turtlesim turtle_teleop_key` on Jetson_Nano to start the keyboard control node. Click the terminal where the control node is started, press the up, down, left, and right arrow keys. If you can control the movement of the turtle on the virtual machine, it means that the configuration is successful.

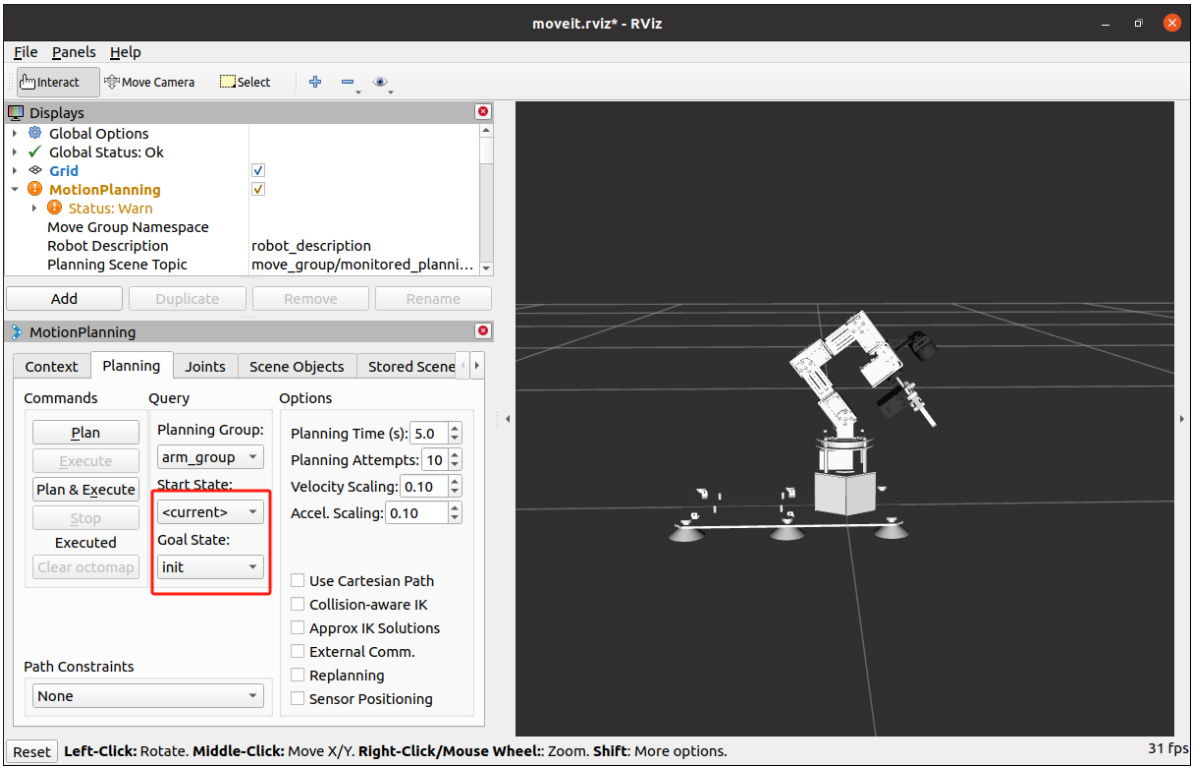


2.2, Start command

Enter the following command in the terminal to start,

```
#Start MoveIt from the virtual machine or Orin motherboard
roslaunch dofbot_pro_config demo.launch
#Start the underlying driver to control the robot arm from the motherboard
rosrun dofbot_pro_info arm_driver.py
#Start the real machine driver program from the virtual machine or Orin
motherboard
rosrun arm_moveit_demo SimulationToMachine.py
```

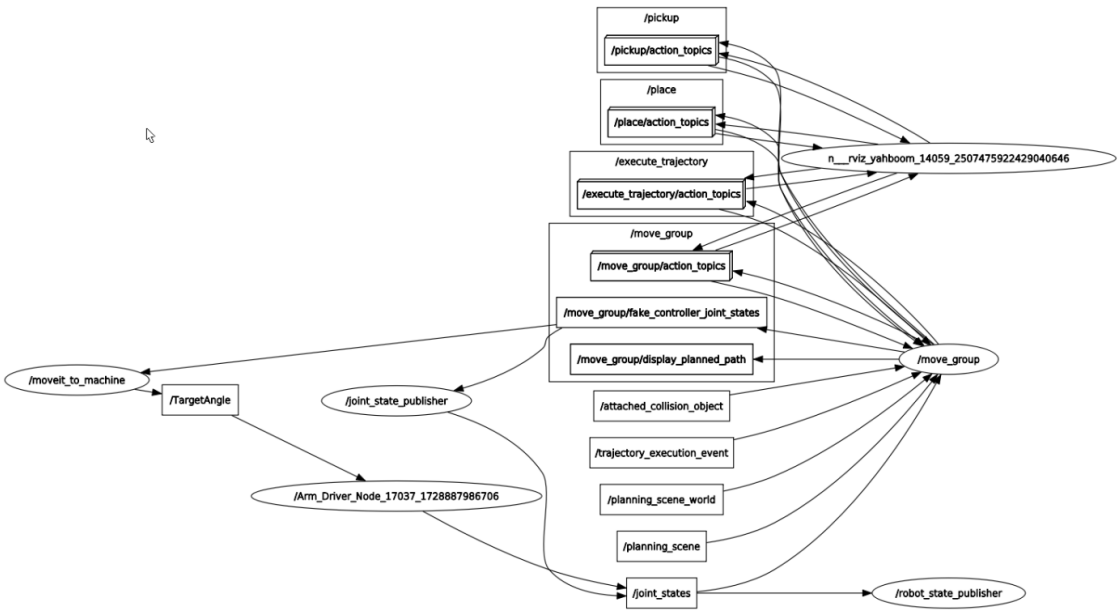
After successfully starting MoveIt, select the [arm_group] planning group, select the target pose of [init], click [Plan&Execute], plan and execute, and the robot arm will follow the robot arm in Rviz to move to the set init pose.



3. View the node communication graph

View the node communication graph through the following command to check the topic communication between nodes,

```
rosrun rqt_graph rqt_graph
```



We can see that the /move_group/fake_controller_joint_states topic published by the /move_group node, the driver real machine node /moveit_to_machine subscribes to the topic information, and after processing, it publishes the /TargetAngle topic message, which is used to control the movement of the robot arm. After the underlying driver node /Arm_Driver_Node receives the topic message, it is processed and published to the underlying layer to control the robot arm to move to the target posture.

4. Core code analysis

Code path:
`/home/jetson/dofbot_pro_ws/src/arm_moveit_demo/scripts/SimulationToMachine.py`

```
#!/usr/bin/env python3
# coding: utf-8
from time import sleep

import rospy
import numpy as np
from math import pi
from yahboomcar_msgs.msg import *
from sensor_msgs.msg import JointState
import sys
np.set_printoptions(threshold=sys.maxsize)

class SimulationToMachine:
    def __init__(self):
        self.joints = [90.0, 90.0, 90.0, 90.0, 90.0, 30.0]
        #Create a subscriber to subscribe to the topic
/move_group/fake_controller_joint_states
        self.subscriber =
rospy.Subscriber("/move_group/fake_controller_joint_states", JointState,
self.topic)
        #Create a publisher to publish the topic /TargetAngle
        self.pub_Arm = rospy.Publisher("TargetAngle", ArmJoint, queue_size=1000)
        sleep(0.1)
        self.pubArm(self.joints)
    #Callback function, mainly for processing the received
/move_group/fake_controller_joint_states topic message data
    def topic(self, msg):
        if not isinstance(msg, JointState): return
        arm_rad = np.array(msg.position)
        DEG2RAD = np.array([180 / pi])
        arm_deg = np.dot(arm_rad.reshape(-1, 1), DEG2RAD)
        if len(msg.position) == 5:
            mid = np.array([90, 90, 90, 90, 90]))
            arm_array = np.array(np.array(arm_deg) + mid)
            for i in range(5): self.joints[i] = arm_array[i]
        elif len(msg.position) == 1:
            arm_array = np.array(np.array(arm_deg) + np.array([180]))
            self.joints[5] = np.interp(arm_array, [90, 180], [30, 180])[0]
        self.pubArm(self.joints)
    #Function to publish/TargetAngle topic message
    def pubArm(self, joints, run_time=1000):
        arm_joint = ArmJoint()
        arm_joint.joints = joints
        arm_joint.run_time = run_time
        self.pub_Arm.publish(arm_joint)
if __name__ == '__main__':
    rospy.init_node("moveit_to_machine")
    SimulationToMachine()
    rate = rospy.Rate(2)
    rospy.spin()
```