# MoveIt anti-kinematics design

## 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 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 tutorials will take running on the Orin motherboard as an example.

## 1. Functional Description

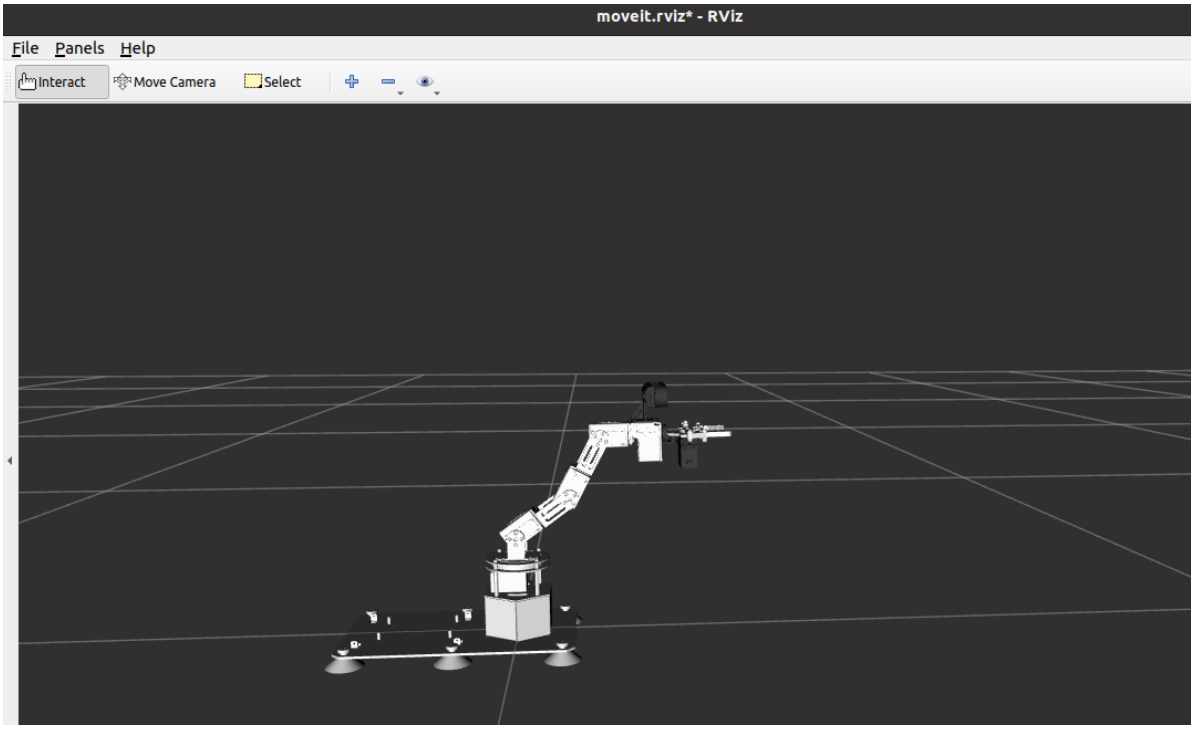
After the program runs, the simulated robotic arm in Rviz will move to the point set by the program.

## 2. Start

```
#Start MoveIt
roslaunch dofbot_pro_config demo.launch
```

Wait for MoveIt to start, then start the anti-kinematics design program, terminal input,

```
rosrun arm_moveit_demo 02_set_pose_plan.py
```




Run as shown in the figure above, the program starts by moving the robot arm to the "down" posture, then plans the path according to the program setting point, calculates the value of each servo, and finally moves to the position set by the program.

## 3. Core code analysis

Code path:
`/home/yahboom/dofbot_ws/src/arm_moveit_demo/scripts/02_set_pose_plan.py`

```
#!/usr/bin/env python3
# coding: utf-8
import rospy
from math import pi
from time import sleep
import moveit_commander
from geometry_msgs.msg import Pose
from moveit_commander.move_group import MoveGroupCommander
from tf.transformations import quaternion_from_euler
# Convert angle to radians
DE2RA = pi / 180

if __name__ == '__main__':
    # Initialize the node
    rospy.init_node("set_pose_py")
    # Initialize the robot arm
    dofbot_pro = MoveGroupCommander("arm_group")
    # Allow replanning after motion planning fails
    dofbot_pro.allow_replanning(True)
    dofbot_pro.set_planning_time(5)
    # Number of planning attempts
    dofbot_pro.set_num_planning_attempts(10)
    # Set the allowable error of position (unit: meter) and attitude (unit:
radian)
    dofbot_pro.set_goal_position_tolerance(0.01)
    dofbot_pro.set_goal_orientation_tolerance(0.01)
    # Set the allowable target error
    dofbot_pro.set_goal_tolerance(0.01)
    # Set the maximum allowed velocity and acceleration
    dofbot_pro.set_max_velocity_scaling_factor(1.0)
    dofbot_pro.set_max_acceleration_scaling_factor(1.0)
    # Set "down" as the target point
    dofbot_pro.set_named_target("down")
    dofbot_pro.go()
    sleep(0.5)
    # Create a pose instance
    pos = Pose()
    # Set the specific position
    pos.position.x = -0.000599999999999989
    pos.position.y = 0.1835691951077983
    pos.position.z = 0.2550887465333698

    pos.orientation.x = 0.017276103846364667
    pos.orientation.y = 7.852827837256886e-05
    pos.orientation.z =  1.3568651988846686e-06
    pos.orientation.w = 0.9998507538964794

    # Set the target point
    dofbot_pro.set_pose_target(pos)
    plan = dofbot_pro.plan()
    #print(plan[1].joint_trajectory)
    #print(plan.joint_trajectory)
```

```
    # Execute multiple times to improve the success rate
    for i in range(5):
        # Motion Planning
        plan = dofbot_pro.plan()
        #print(plan.joint_trajectory.points)
        if len(plan[1].joint_trajectory.points) != 0:
            print ("plan success")
            # Run after successful planning
            dofbot_pro.execute(plan[1])
            break
        else: print ("plan error")
    moveit_commander.roscpp_shutdown()
    moveit_commander.os._exit(0)
```