4. MoveIt kinematics design

This lesson takes MoveIT simulation as an example. If you need to set up the real machine and simulation to be synchronized, please see the lesson [02, MoveIt Precautions and Controlling the Real Machine]. ! ! ! be safe! ! !

The effect demonstration is a virtual machine and other main control running conditions (related to the main control performance, depending on the actual situation).

4.1. Position description in three-dimensional space

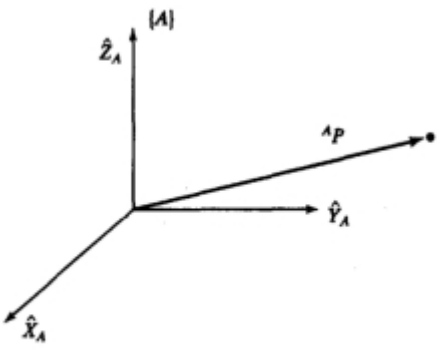
First, a coordinate system is specified. Relative to the coordinate system, the position of the point can be represented by a 3-dimensional column vector; the orientation of the rigid body can be represented by a 3×3 rotation matrix. The 4×4 homogeneous transformation matrix can unify the description of rigid body position and attitude (pose). It has the following advantages:

(1) It can describe the pose of the rigid body and the relative pose (description) of the coordinate system.

(2) It can represent the transformation (mapping) of a point from the description of one coordinate system to the description of another coordinate system.

(3) It can represent the transformation (operator) of the pose description before and after the rigid body motion.

- Position description - position vector

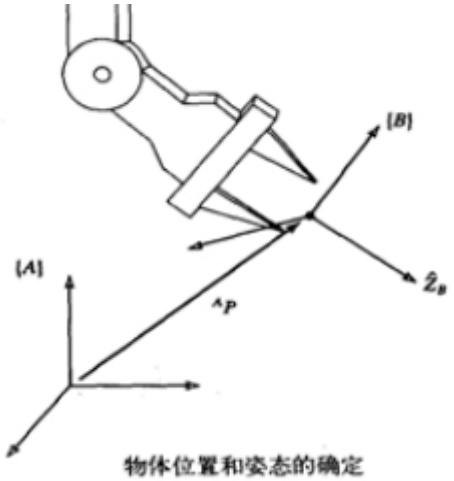


相对于坐标系的矢量（示例）

A coordinate system {A} is represented by three mutually orthogonal unit vectors with arrows. Then the spatial position of point p in the coordinate system {A} is expressed as:

$$^{A}P = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

- Description of orientation - rotation matrix



物体位置和姿态的确定

Commonly used rotation transformation matrices are to rotate an angle around the X-axis, around the Y-axis, or around the Z-axis. they are, respectively

$$R(X,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} R(Y,\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} R(Z,\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Non-homogeneous expression

After the vector a undergoes one rotation R and one translation t, we get $a': a'=R*a+t$

- Homogeneous expression

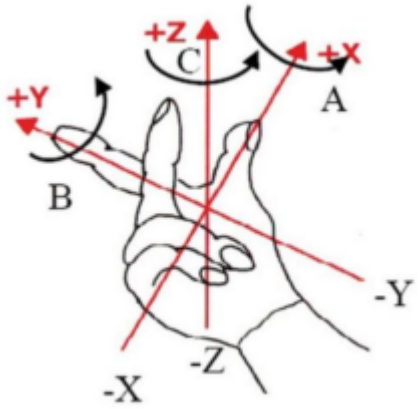
$$\begin{bmatrix} a' \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} = T \begin{bmatrix} a \\ 1 \end{bmatrix}$$

The general rotation matrix R is:

$$R = R_z(\beta) R_y(\alpha) R_x(\theta)$$

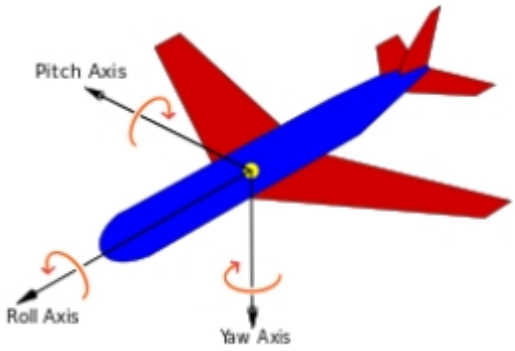
The rotation matrix rotation process here first rotates the θ angle around the X-axis, then rotates the α angle around the Y-axis, and finally rotates the β angle around the Z-axis.

- Right hand rule



The thumb points to X, the index finger extends to Y, and the middle finger points to Z.

- RPY and Euler angles



The rotation order according to its own coordinate system is Z-->Y-->X, which is called eulerYPR (Yaw Pitch Roll);

The rotation sequence according to the external coordinate system (reference coordinate system) is x-->y-->z, which is called RPY (Roll Pitch Yaw);

When describing the same posture, the above two expressions are equivalent, that is, the angle value of Yaw Pitch Roll is the same.

Definition: Yaw yaw angle Ƴ; Pitch pitch angle β; Roll roll angle α

- Quaternion method

The rough definition is that a vector is represented by a scalar and three imaginary axes, which is called a quaternion. The range of each data [x, y, z, w] is [-1,1].

4.2. Start

Correct solution: In a robotic arm, each joint angle is known and the end pose is solved.

Inverse solution: In a robotic arm, with the end position (except the gripper) known, find the angle of each joint.

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

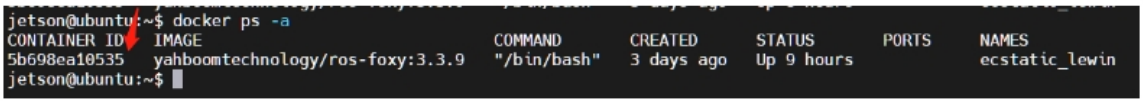
Start MoveIT

```
roslaunch arm_moveit_demo x3plus_moveit_demo.launch sim:=true
```

<PI5 needs to open another terminal to enter the same docker container

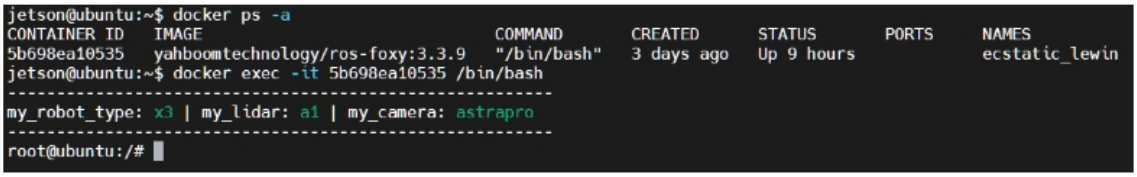
1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

```
docker ps -a
```



2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```



After successfully entering the container, you can open countless terminals to enter the container.

Start the inverse solution node (choose one of the two)

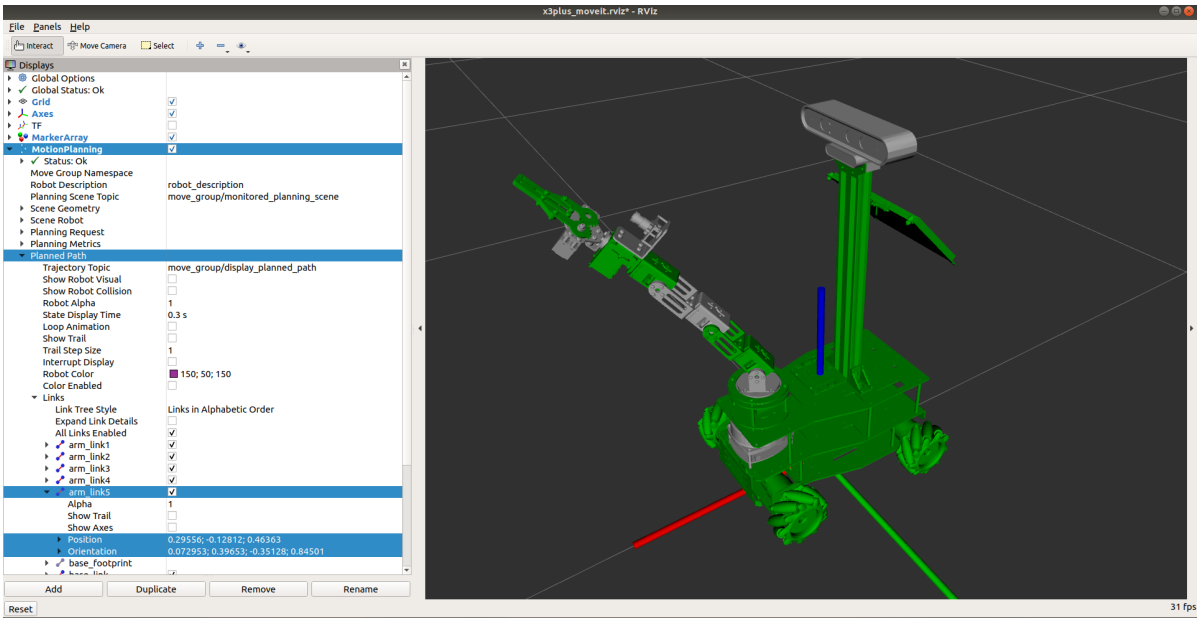
```
rosrun arm_moveit_demo 02_set_pose_plan # C++
rosrun arm_moveit_demo 02_set_pose_plan.py # python
```

Start the correct solution node (choose one of the two)

```
rosrun arm_moveit_demo 03_set_joint_plan # C++
rosrun arm_moveit_demo 03_set_joint_plan.py # python
```

4.3. Reverse the key code

Note: Due to the range limitation of the manipulator, it is possible that the specified target pose cannot be reached. Repeated execution of the cycle can only slightly improve the planning success rate. Not only the position must be considered, but also the attitude of the end. We can view the pose of each link through rviz, as shown below,



[Position] and [Orientation] in [arm_link5] in the above figure are the end poses. When using the inverse solution, you can also directly set the position and orientation quaternions of the target point.

4.3.1, python code

Set target pose

```
    #Create pose instance
    pos = PoseStamped()
    # Set location
    pos.header.frame_id = "base_footprint"
    pos.header.stamp = rospy.Time.now()
    pos.pose.position.x = 0.438534
    pos.pose.position.y = 0.00145727
    pos.pose.position.z = 0.224613
    # Set pose
    pos.pose.orientation.x = -0.00450228
    pos.pose.orientation.y = 0.702852
    pos.pose.orientation.z = -0.001513
    pos.pose.orientation.w = 0.71132
```

Set target point

```
    # Set target point
    yahboomcar.set_pose_target(pos,end_effector_link)#end_effector_link is the
link of the terminal, here is arm_link5
```

Loop execution path planning

```
    # Execute multiple times to improve success rate
    for i in range(5):
        #motion planning
        plan = yahboomcar.plan()
        if len(plan.joint_trajectory.points) != 0:
            print("plan success")
            #Run after successful planning
            yahboomcar.execute(plan)
            break
        else: print ("plan error")
```

4.3.2, C++ code

Set target pose

```
    //Set specific location
    geometry_msgs::Pose pose;
    pose.position.x = 0.2;
    pose.position.y = 0;
    pose.position.z = 0.2178;
    //Set target posture
    tf::Quaternion quaternion;
    //The unit of RPY is the angle value
    double Roll = -180;
    double pitch = 45;
    double Yaw = -180;
```

Convert the pose to quaternion and set the target point

```
    // RPY to quaternion
    quaternion.setRPY(Roll * DE2RA, Pitch * DE2RA, Yaw * DE2RA);
    pose.orientation.x = quaternion.x();
    pose.orientation.y = quaternion.y();
    pose.orientation.z = quaternion.z();
    pose.orientation.w = quaternion.w();
    yahboomcar.setPoseTarget(pose);
```

Loop execution path planning

```
    int index = 0;
    // Execute multiple times to improve success rate
    while (index <= 10) {
        moveit::planning_interface::MoveGroupInterface::Plan plan;
        // motion planning
        const moveit::planning_interface::MoveItErrorCode &code =
yahboomcar.plan(plan);
        if (code == code.SUCCESS) {
            ROS_INFO_STREAM("plan success");
            yahboomcar.execute(plan);
            break;
        } else {
            ROS_INFO_STREAM("plan error");
        }
```

```
            index++;
        }
```

4.4. Correct key code

4.4.1, python code

Set target point

```
joints = [0, 0.79, -1.57, -1.57, 0]
yahboomcar.set_joint_value_target(joints)
```

Loop execution path planning

```
# Execute multiple times to improve success rate
    for i in range(5):
        #motion planning
        plan = yahboomcar.plan()
        if len(plan.joint_trajectory.points) != 0:
            print("plan success")
            #Run after successful planning
            yahboomcar.execute(plan)
            break
        else:
            print("plan error")
```

4.4.2, C++ code

Set target point

```
    //Set specific location
    vector<double> pose{0, 0.79, -1.57, -1.57, 0};
    yahboomcar.setJointValueTarget(pose);
```

Loop execution path planning

```
    moveit::planning_interface::MoveGroupInterface::Plan plan;
    const moveit::planning_interface::MoveItErrorCode &code =
yahboomcar.plan(plan);
    if (code == code.SUCCESS) {
        ROS_INFO_STREAM("plan success");
        //display trajectory
        string frame = yahboomcar.getPlanningFrame();
        moveit_visual_tools::MoveItVisualTools tool(frame);
        tool.deleteAllMarkers();
        tool.publishTrajectoryLine(plan.trajectory_,
yahboomcar.getCurrentState()->getJointModelGroup("arm_group"));
        tool.trigger();
        yahboomcar.execute(plan);
    } else {ROS_INFO_STREAM("plan error");
    }
```

4.5, MoveIT kinematics plug-in

The MoveIT kinematics plug-in is the file [kinematics.yaml]

If you need to replace or modify, just modify the [kinematics.yaml] file directly.

```
roscd x3plus_moveit_config/config
sudo vim kinematics.yaml
```

- kdl

```
arm_group:
   kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
   kinematics_solver_search_resolution: 0.005
   kinematics_solver_timeout: 0.005
```

- trac_ik

```
arm_group:
   kinematics_solver: trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin
   kinematics_solver_search_resolution: 0.005
   kinematics_solver_timeout: 0.005
```