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

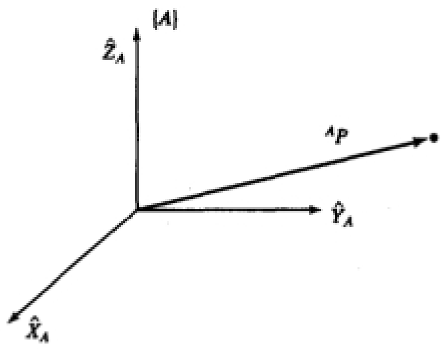
First, a coordinate system is specified. Relative to the coordinate system, the position of the point can be represented by a 3-dimensional column vector; the orientation of the rigid body can be represented by a 3×3 rotation matrix. The 4×4 homogeneous transformation matrix can unify the description 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.

- Location description - location vector

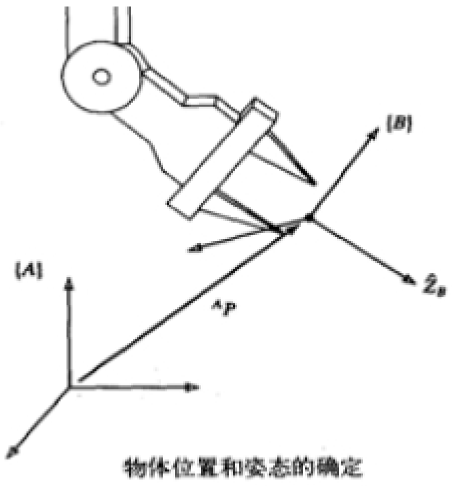


相对于坐标系的矢量（示例）

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



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

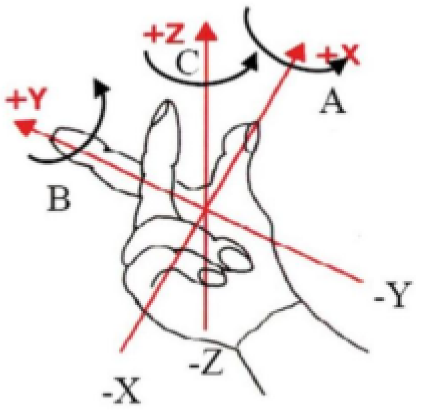
$$\begin{bmatrix} a\prime \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} = T \begin{bmatrix} a \\ 1 \end{bmatrix}$$

The common rotation matrix R is:

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

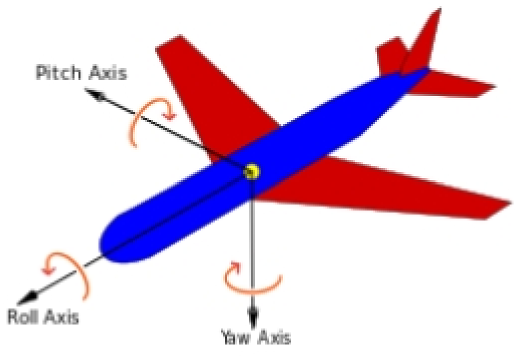
The rotation matrix rotation process here first rotates the θ angle around the X-axis, then rotates the α angle around the Y-axis, and finally rotates the β angle around the Z-axis.

- Right hand rule



The thumb points to X, the index finger reaches Y, and the middle finger is in the direction of 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 Υ; 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 up

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.

Start MoveIT

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

Start the anti-solution node

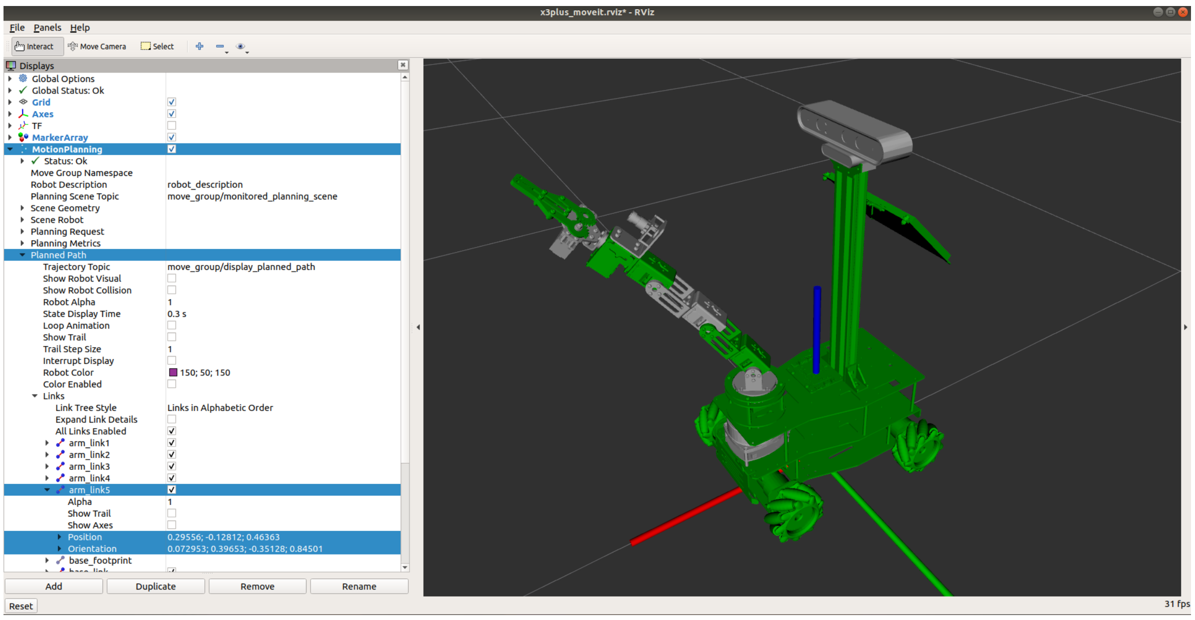
```
rosrun arm_moveit_demo 02_set_pose_plan
```

Start the correct solution node

```
rosrun arm_moveit_demo 03_set_joint_plan
```

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,



Global Options	
Global Status: Ok	
Grid	✓
Axes	✓
TF	☐
MarkerArray	✓
MotionPlanning	✓
Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planning_scene
Scene Geometry	
Scene Robot	
Planning Request	
Planning Metrics	
Planned Path	
Trajectory Topic	move_group/display_planned_path
Show Robot Visual	☐
Show Robot Collision	☐
Robot Alpha	1
State Display Time	0.3 s
Loop Animation	☐
Show Trail	☐
Trail Step Size	1
Interrupt Display	☐
Robot Color	🟪 150; 50; 150
Color Enabled	☐
Links	
Link Tree Style	Links in Alphabetic Order
Expand Link Details	☐
All Links Enabled	✓
arm_link1	✓
arm_link2	✓
arm_link3	✓
arm_link4	✓
arm_link5	✓
Alpha	1
Show Trail	☐
Show Axes	☐
Position	0.29556; -0.12812; 0.46363
Orientation	0.072953; 0.39653; -0.35128; 0.84501
base_footprint	

[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.C++ code

Set target pose

```
    //Set specific location
    geometry_msgs::Pose pose;
    pose.position.x = 0.18721125717113798;
    pose.position.y = -0.008718652395814977;
    pose.position.z = 0.4787351295417709;
    // Set target posture
    tf::Quaternion quaternion;
    //The unit of RPY is the angle value
    //double Roll = -180;
    //double Pitch = 45;
    //double Yaw = -180;
    // RPY to quaternion
    //quaternion.setRPY(Roll * DE2RA, Pitch * DE2RA, Yaw * DE2RA);
    pose.orientation.x = 0.08637313729188083;
    pose.orientation.y = 0.40819551903689105;
    pose.orientation.z = 0.10050372779920543;
    pose.orientation.w = 0.9032248336328139;
    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;
        // Movement 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. Positive solution key code

4.4.1. 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");
        // show 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
```