4.Kinematics design

1. Position description in three-dimensional space

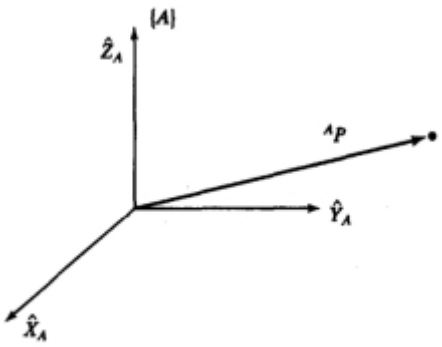
First, a coordinate system is specified. Relative to the coordinate system, the position of the point can be represented by a 3-dimensional column vector; the orientation of the rigid body can be represented by a 3×3 rotation matrix. The 4×4 homogeneous transformation matrix can unify the description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

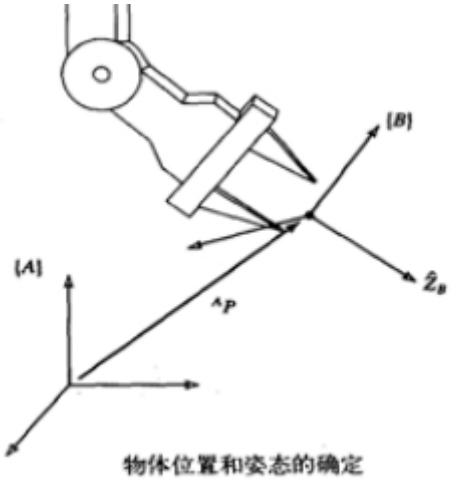


相对于坐标系的矢量（示例）

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:

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

- 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.

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

- Non-homogeneous expression

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

- Homogeneous expression

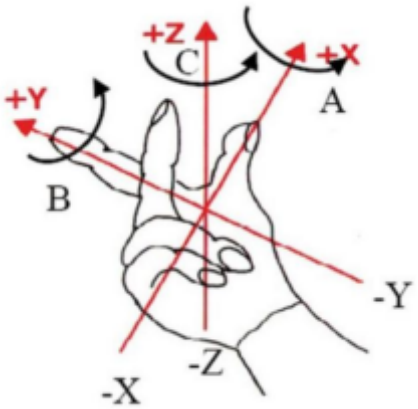
$$\begin{bmatrix}a\prime\\ 1\end{bmatrix}=\begin{bmatrix}R & t\\ 0^T & 1\end{bmatrix}\begin{bmatrix}a\\ 1\end{bmatrix}=T\begin{bmatrix}a\\ 1\end{bmatrix}$$

The general rotation matrix R is:

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

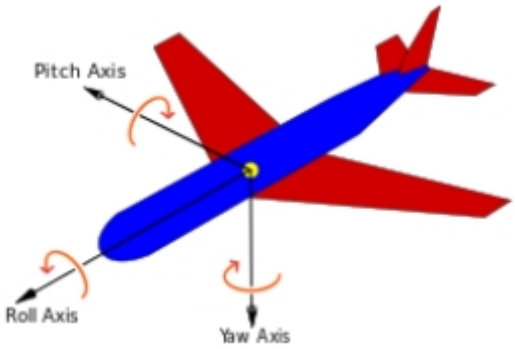
The rotation matrix rotation process here first rotates around the X-axis by θ angle, then rotates around Y-axis by α angle, and finally rotates around Z-axis by β angle.

- 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].

2. Start robotic arm simulation

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

- **Starts Virtual machine simulation**

```
roslaunch dofbot_config demo.launch
```

- **Start kinematics design node**

```
rosrun arm_moveit_demo 02_set_pose_plan
```

Close case: [ctrl+c] to close. If it cannot be closed, execute [ctrl+z] again.

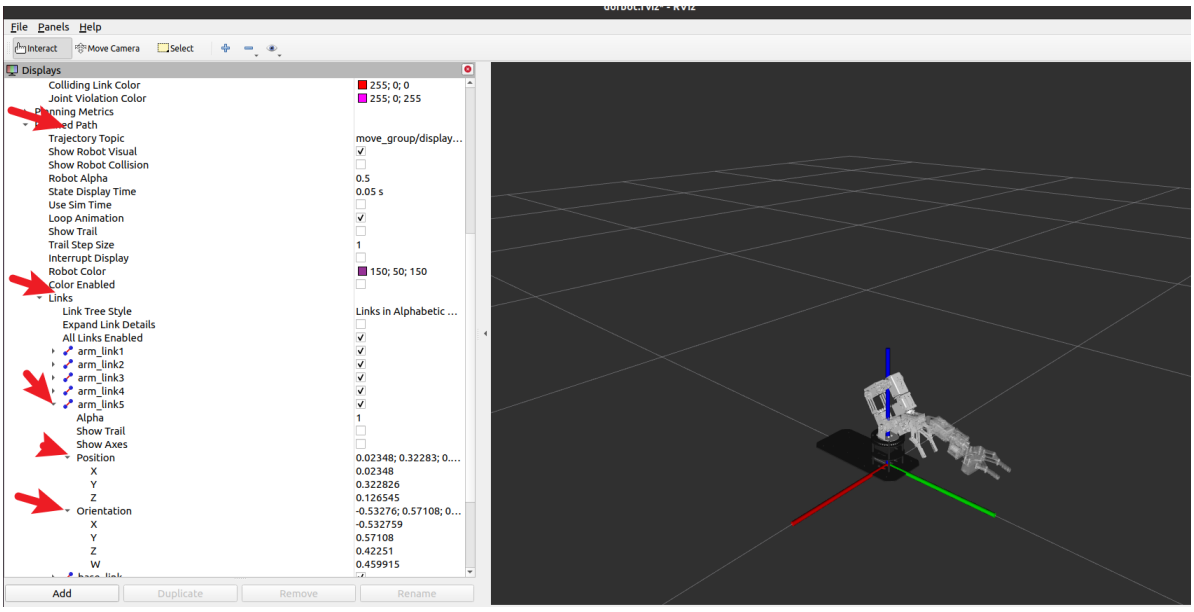
As shown in the figure, follow the red arrow on the left to view the end pose information (position and attitude) in real time.

Position: position;

Orientation: four elements representing attitude.

When planning motion, not only the position reached by the robotic arm must be considered, but also the attitude when it reaches that position. Multiple planning can improve the success rate, and you can also consider replacing the kinematics plug-in.

Sample image, as shown below.



- Code design

Create pose instance

```
pos = Pose()
```

Set pose information

```
# Set specific location
    pos.position.x = 0.008583
    pos.position.y = 0.124503
    pos.position.z = 0.088820
    # The unit of RPY is the angle value
    # The unit of RPY is the angle value
    roll = -140.0
    pitch = 0.0
    yaw = 0.0
```

Add target point

```
dofbot.set_pose_target(pos)
```

Execution plan

```
plan = dofbot.plan()
```

Code path: dofbot_ws/src/dofbot_moveit/scripts/02_motion_plan.py

Experimental phenomenon: You can see that the robotic arm in rviz will move to the position we set.

3. Start the real machine

Robotic arm rviz simulation movement + real machine operation

Close previous program and open another terminal to enter the command (this program is simulated in rviz plus real machine motion)

```
cd dofbot_ws/
source devel/setup.bash
rosrun dofbot_moveit 2.py # python file
```

Close case: [ctrl+c] to close. If it cannot be closed, execute [ctrl+z] again.

Key part of the program code description:

```
#Subscribe to the angle of each joint published by moveIt
subscriber = rospy.Subscriber("/joint_states", JointState, topic)
#The subscribed angle is calculated and sent to the python driver library
for i in range(6):
        joints[i] = (msg.position[i] * RA2DE) + 90
        if(i == 5):
            joints[i] = (msg.position[i] * 116) + 180
##Call driver function
sbus.Arm_serial_servo_write6_array(joints, 100)
```

Code path: dofbot_ws/src/dofbot_moveit/scripts/2.py

Experimental phenomenon: The real robot arm will move along with the movement of the model in rviz.

The terminal will print planning success.

The real robotic arm moves to the planned position.

3. MoveIT kinematics plugin

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
```