4. MoveIt kinematics design

4. MoveIt kinematics design

- 4.1. Pose description of 3D space
- 4.2. Start
- 4.3. Reverse solution key code
  - 4.3.1.python code
  - 4.3.2.C++code
- 4.4. Positive solution key code
  - 4.4.1.python code
  - 4.4.2.C++code
- 4.5. MoveIT kinematics plugin

This lesson takes the MoveIT simulation as an example. If you need to set the synchronization between the real machine and the simulation, please refer to the lesson [02, MoveIt Precautions and Controlling the Real Machine]. ! ! ! be careful! ! !

The effect demonstration is a virtual machine, and other masters are running (related to the performance of the master, depending on the actual situation).

4.1. Pose description of 3D space

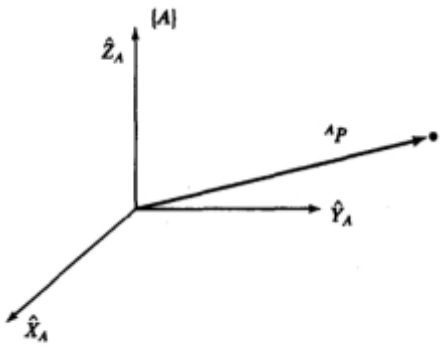
First, a coordinate system is specified, relative to the coordinate system, the position of the point can be represented by a 3-dimensional column vector; the orientation of the rigid body can be represented by a 3×3 rotation matrix. The 4×4 homogeneous transformation matrix can unify the description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

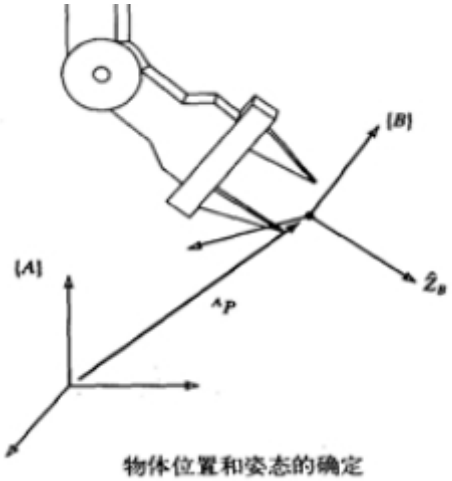


相对于坐标系的矢量（示例）

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

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

- Description of the orientation - rotation matrix



物体位置和姿态的确定

The commonly used rotation transformation matrix is to rotate an angle around the X axis, around the Y axis or around the Z axis. they are respectively:

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

- Inhomogeneous representation

After the vector a has undergone a rotation R and a translation t,get $a' : a'=R*a+t$

- Homogeneous representation

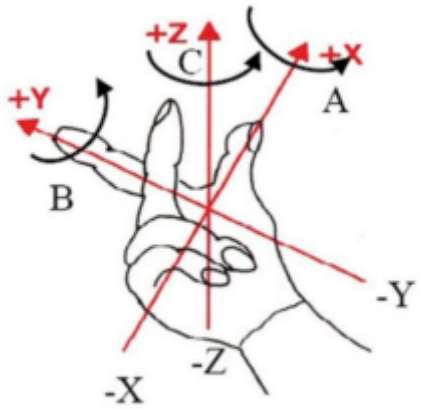
$$\begin{bmatrix} a\prime \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} = T \begin{bmatrix} a \\ 1 \end{bmatrix}$$

The universal rotation matrix R is:

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

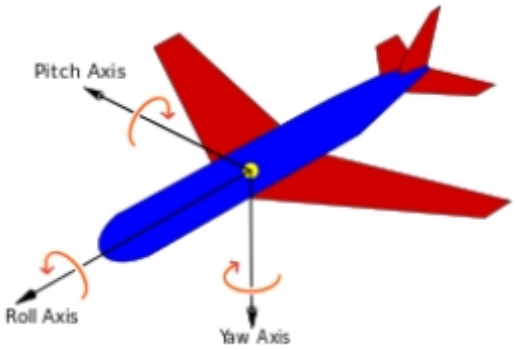
The rotation matrix rotation process here is to rotate the angle θ around the X axis first, then rotate the angle α around the Y axis, and finally rotate the angle β around the Z axis.

- Right-hand rule



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

- RPY and Euler Angles



According to the rotation order of its own coordinate system, it is Z-->Y-->X, which is called eulerYPR (Yaw Pitch Roll);

According to the rotation order of the external coordinate system (reference coordinate system) is x-->y-->z,which is called RPY (Roll Pitch Yaw);

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

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

- Quaternion method

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

4.2. Start

Positive solution: In a robotic arm,knowing each joint angle, solve for the end pose.

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

Start the MoveIT

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

Start the inverse solution node (choose one of two)

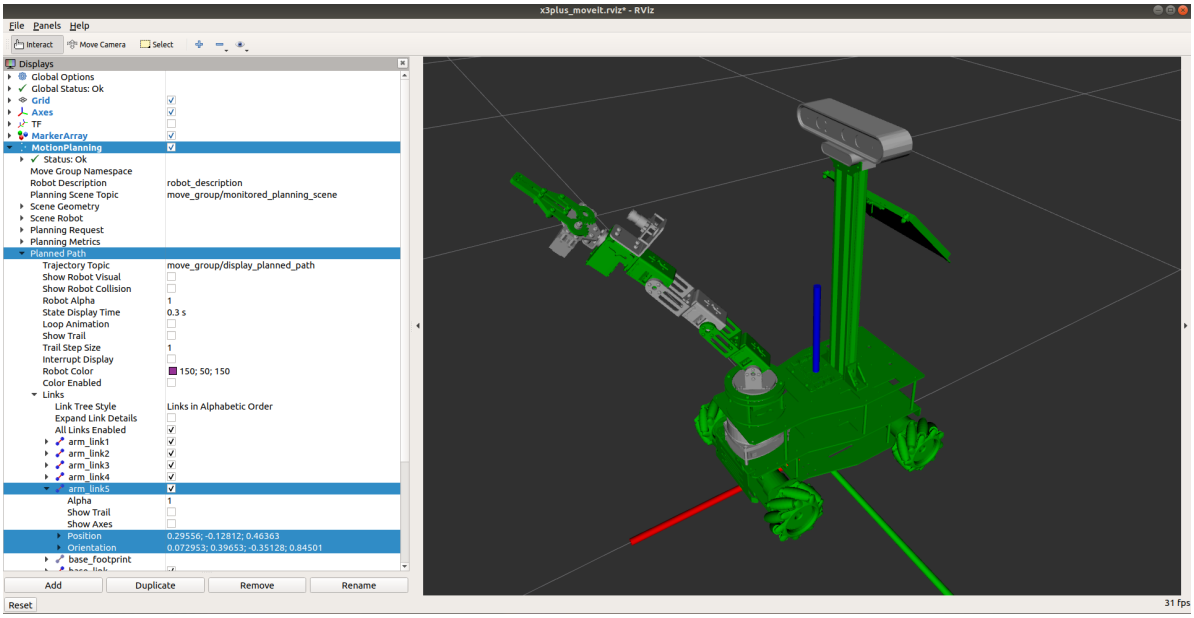
```
rosrun arm_moveit_demo 02_set_pose_plan      # C++
rosrun arm_moveit_demo 02_set_pose_plan.py   # python
```

Start the positive solution node (choose one of two)

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

4.3. Reverse solution key code

Note: Due to the limited range of the robotic arm, it is possible that the specified target pose cannot be reached, and multiple executions of the loop can only slightly improve the planning success rate. Consider not only the position, but also the pose of the end. We can view the pose of each link through rviz, as shown below,



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

4.3.1.python code

Set target pose

```
    # Create the 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_linkis the
link of the terminal here is arm_link5
```

Loop execution path planning

```
    # Execute multiple times to improve the success rate
    for i in range(5):
        # Exercise planning
        plan = yahboomcar.plan()
        if len(plan.joint_trajectory.points) != 0:
            print ("plan success")
            #  Run after planning is successful
            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 pose
    tf::Quaternion quaternion;
    //The unit of RPY is the angle value
    double Roll = -180;
    double Pitch = 45;
    double Yaw = -180;
```

Convert pose to quaternion, set 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 the success rate
    while (index <= 10) {
        moveit::planning_interface::MoveGroupInterface::Plan plan;
        // Exercise 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.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 the success rate
    for i in range(5):
        #Exercise planning
        plan = yahboomcar.plan()
        if len(plan.joint_trajectory.points) != 0:
            print ("plan success")
            #Run after planning is successful
            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");
        //Show track
        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 plugin

The MoveIT kinematics plugin is the file [kinematics.yaml]

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

```
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
```