Journal of Electronic Research and Application, 2026, Volume 10, Issue 1
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

# Research on Motion Simulation of Panda Manipulator Based on ROS2

**Boru Wang*, Wei Liu**

Jilin University of Chemical Technology, Ji'lin, Jilin, China

**Author to whom correspondence should be addressed.*



**Abstract:** Focusing on the research issues of path optimization and collision avoidance in robotic arm motion planning, this paper will construct a high-fidelity simulation environment using the ROS2 framework. By integrating MoveIt, Rviz visualization tool packages, and the Panda robotic arm URDF file, along with leveraging the DDS communication mechanism of ROS2, low-latency data interaction between the planning module and the simulation environment is achieved. The upper computer software is developed to conduct simulation studies on the path planning and trajectory interpolation principles of the robotic arm, thereby verifying the reliability of the ROS2 distributed architecture in robotic arm simulation.

**Keywords:** ROS2; Robotic arm; Path optimization; Trajectory interpolation

**Online publication:** February 12, 2026

## 1. Introduction

With the development of science and technology and the improvement of people's living standards, robotic arms are being increasingly widely applied in various fields, and users have also set higher performance requirements for them [1]. To meet the ever-more complex application demands, it is necessary to conduct various simulations during the development of the robotic arm body to save R&D costs and enhance R&D efficiency. Currently, the mainstream simulation platforms for developing robotic arms include Matlab, TeamBots, CARMEN, ROS, and so on. Among them, ROS2 is more widely utilized by universities, research institutions, and enterprises. ROS2 is the second-generation robot operating system specifically designed for modern robotic systems. It employs a DDS-based (Data Distribution Service) peer-to-peer (P2P) communication mechanism, supports task responses at the millisecond level, and has achieved significant improvements in real-time performance, security, cross-platform compatibility, and distributed architecture, making it an ideal choice for robot development ranging from consumer-grade to industrial-grade applications [2–4].

## 2. ROS system-related tools

### 2.1. Configuration of the robotic arm

Before controlling the robotic arm, it is necessary to use the MoveIt Setup Assistant to configure the robotic arm's description file in order to generate the required YAML files and various launch files for controlling its motion. This article utilizes the built-in Panda robotic arm URDF file provided by ROS. The file describes the parameters of the Panda robotic arm, such as the fact that it has seven joints, the installation positions of the motors on the links, and the motion angle and speed limitations of the joints, among other details. The steps for importing the URDF file into the MoveIt Setup Assistant to generate the configuration package are illustrated in **Figure 1**.

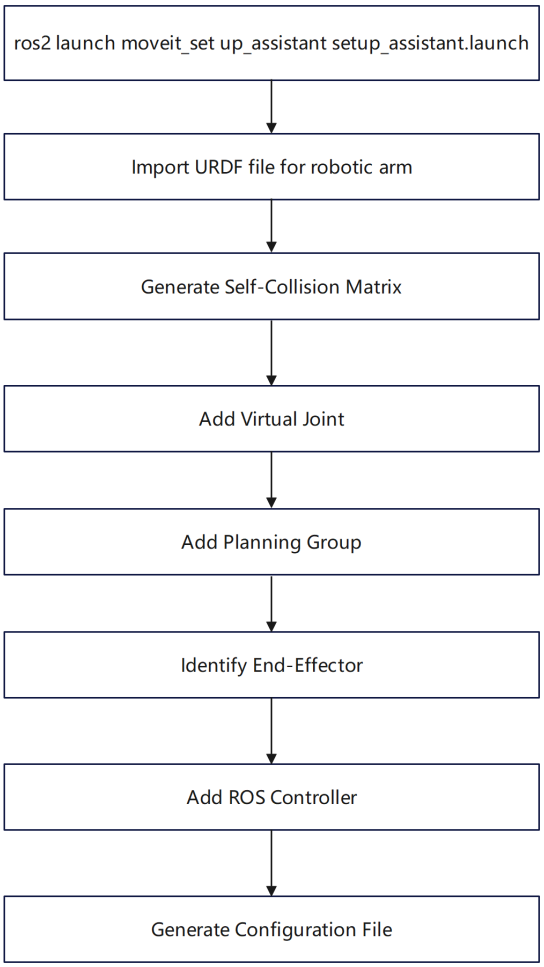


**Figure 1.** Configuration process for the panda robotic arm.

During the configuration process, the KDL algorithm library can be selected to solve the forward and inverse kinematics of the robotic arm. The generated configuration package includes a config folder and a launch folder. The config folder contains a series of YAML files that describe the initial position of the robotic arm, joint limits, kinematics algorithm type, as well as MoveIt controllers and ROS controllers [5]. The descriptive information in these files is closely related to the motion state of the robotic arm.

### 2.2. MoveIt system framework

MoveIt plays a highly significant role in the motion control process of robotic arms. Its functional block diagram

is illustrated in **Figure 2**. The move_group node retrieves the URDF and SRDF files of the robotic arm from the parameter server [6]. It interacts with the robot controller via the JointTrajectoryAction interface. Meanwhile, users can configure MoveIt through the interfaces provided by MoveIt.

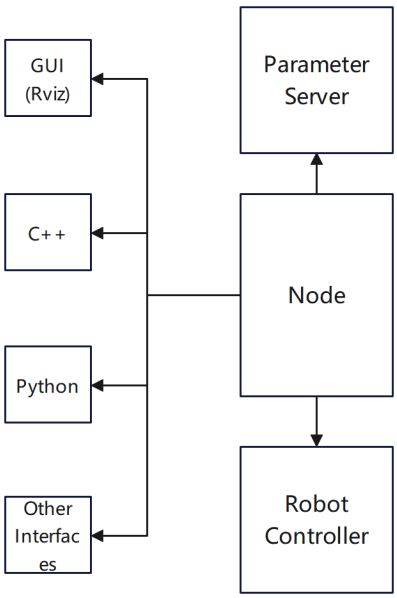


**Figure 2.** Block diagram of the MoveIt system.

## 2.3. Rviz visualization

Rviz is a 3D visualization tool capable of real-time displaying the joint states and motion trajectories of a robotic arm. Additionally, it allows for the control of the robotic arm through its built-in plugins [7]. By running a launch file and observing the three-dimensional state of the robotic arm in Rviz, it can be seen that after running the demo file, the state of the robotic arm in Rviz is as shown in **Figure 3**, where all joint angles of the robotic arm are 0, and the Roll-Pitch-Yaw (RPY) angle values are 3.14, 0, and 0, respectively.

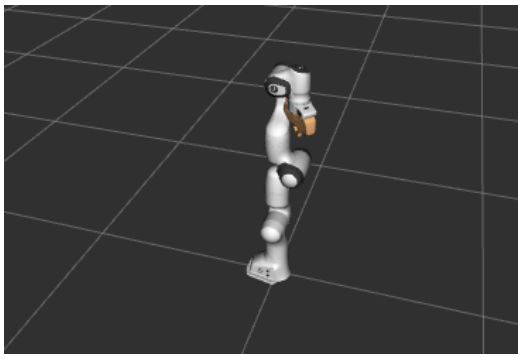

**Figure 3.** Panda robotic arm.

# 3. Route planning algorithm

## 3.1. Principles of the RRT algorithm and the RRT* algorithm

The RRT algorithm, whose full name is Rapidly-exploring Random Tree algorithm, is a sampling-based path planning algorithm [8]. Its basic process is illustrated in **Figure 4**. It incrementally constructs a tree that extends

from the starting point to the target region, expanding one node at a time in an incremental manner. It gradually explores the space until the target point or target region is found. The experimental simulation is shown in **Figure 5**.

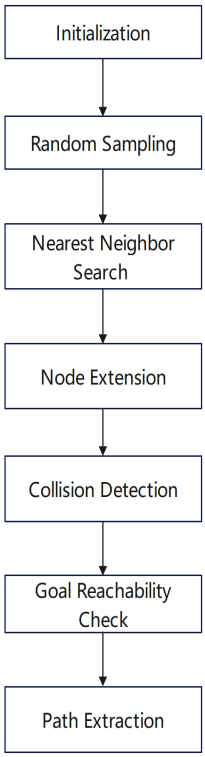


**Figure 4.** Flowchart of the RRT algorithm.

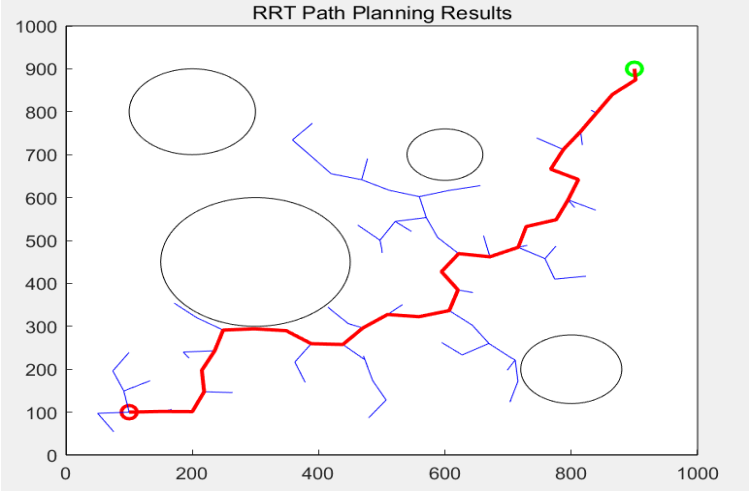


**Figure 5.** Simulation results of the RRT algorithm.

The RRT* algorithm is an improved version of the RRT algorithm, with its primary enhancement being the introduction of a path optimization mechanism. This mechanism ensures that a feasible path is found while further optimizing the path quality, such as by identifying shorter or more optimal paths [9]. It achieves this by reselecting parent nodes and rewiring the random tree, thereby reducing path costs. This is the main distinction between the RRT* algorithm and the RRT algorithm. The experimental simulation is illustrated in **Figure 6**.

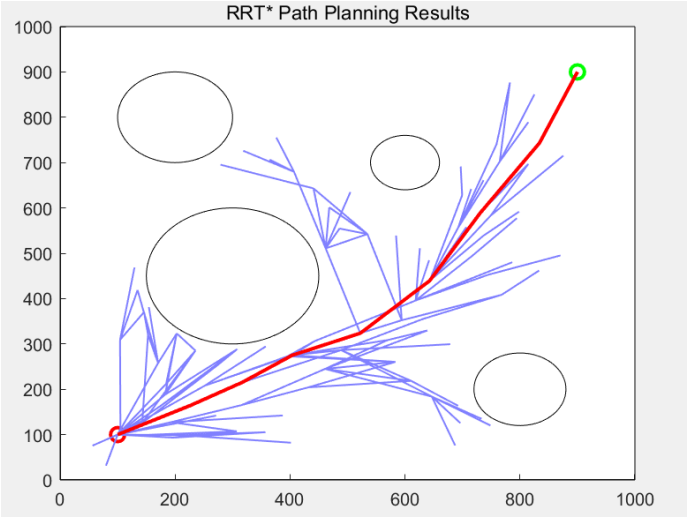


**Figure 6.** Simulation results of the RRT* algorithm.

## 3.2. Comparative testing of RRT and RRT* algorithms

On the upper computer, select the RRT and RRT* algorithms respectively, with the target point set to (0.5, 0.3, 0.4) for both. Click the "Add obstacles" button to introduce obstacles, where the length, width, and height of the obstacles are set to 0.5, 0.05, and 0.5, respectively, as shown in **Figure 7**. Then, set the upper limit for the planning time, click execute to conduct repeated experiments, and record the experimental data obtained, as depicted in **Figure 8**.

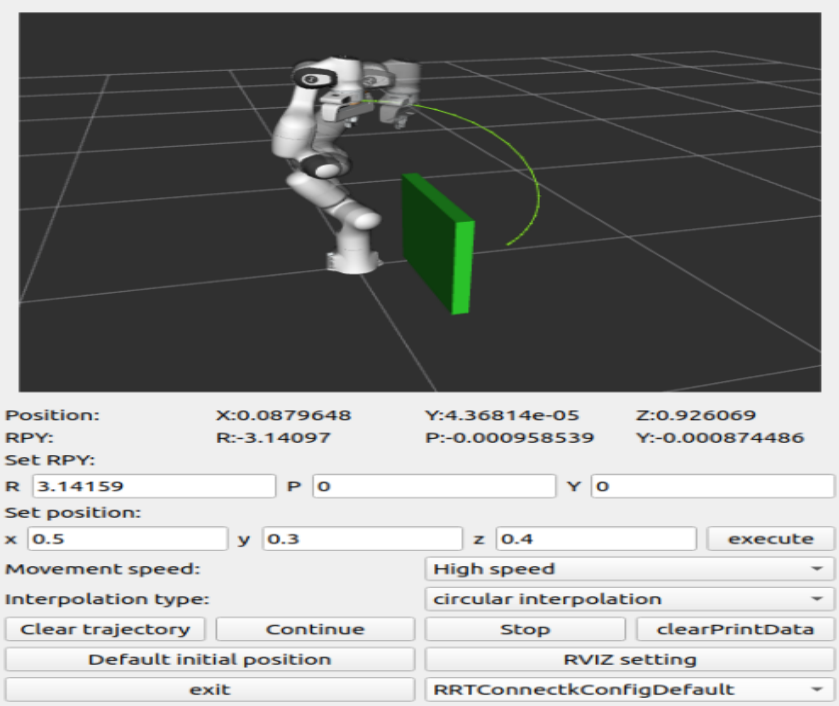


**Figure 7.** Upper computer interface.

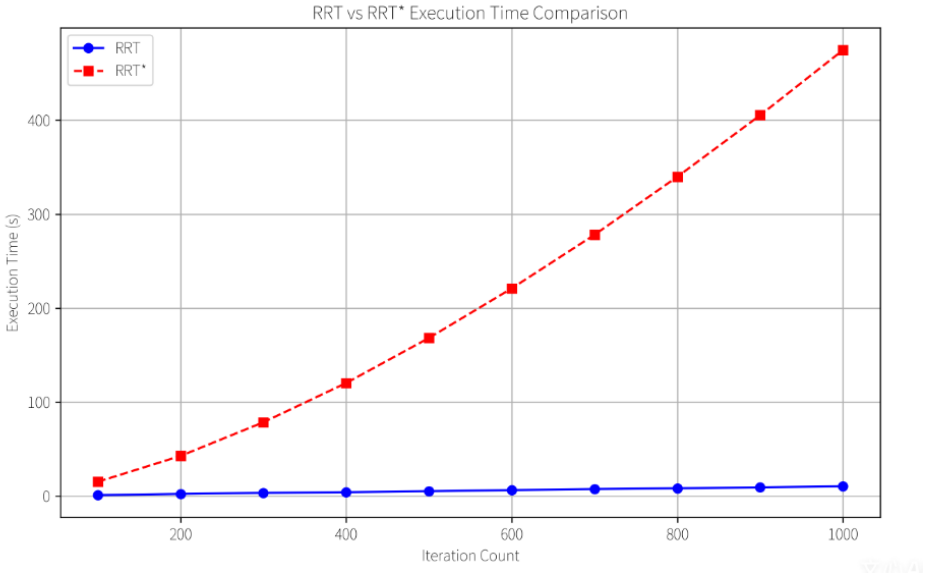


**Figure 8.** Comparative diagram of algorithm execution.

In the experiments, the RRT algorithm can always find a path from the starting point to the destination as long as it is given a sufficiently long planning time. However, due to its random sampling nature, the path it generates is often not optimal and exhibits low search efficiency. When the upper limit of the planning time is reduced, the RRT algorithm has a relatively high probability of failing to plan a path. In contrast, the RRT* algorithm demonstrates a significant improvement in search efficiency during the experiments, with a notably shorter planning time and relatively stable fluctuations in repeated planning durations.

## 4. Interpolation trajectory planning

When using the Configuration Assistant to generate a robotic arm configuration package, the RRT* algorithm from the OMPL motion planning library is selected as the path planning algorithm. After setting the destination coordinates, the planner will generate a path. However, this path only represents a route from the starting point to the destination and does not include the motion parameters for each joint of the robotic arm. Therefore, after the path is planned, it is necessary to perform interpolation on the path. Once the coordinates of each interpolation point are calculated, these coordinate data are transmitted to the inverse kinematics solver module to determine the velocity, acceleration, and other motion parameters of each joint at every point. Thus, calculating a series of suitable interpolation points is crucial for the smooth operation of the robotic arm [10]. After setting the destination coordinates and RPY (Roll, Pitch, Yaw) angles in the upper computer software and clicking the execute button, the robotic arm will plan a path. To achieve smooth motion of the robotic arm's end-effector along the planned path, linear interpolation or circular interpolation is typically required.

### 4.1. Principles of linear interpolation

Given the starting point coordinates as ($x_1$, $y_1$, $z_1$) and the destination point coordinates as ($x_2$, $y_2$, $z_2$), the straight-line distance between the two points in space is calculated as follows:

$$L = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{1}$$

The number of interpolation points is:

$$N = \begin{cases} \frac{L}{d}, \frac{L}{d}(Integer) \\ int\left(\frac{L}{d}\right), \frac{L}{d}(Non - integer) \end{cases} \tag{2}$$

Where  represents the timing step length, $v$ is is the end-effector's motion velocity, and $t_n$ denotes the total timing interpolation duration. The average incremental displacement along the x, y, and z axes for each interpolation point, relative to the previous one, is given by:

$$\Delta x = (x_2 - x_1)/N \tag{3}$$

$$\Delta y = (y_2 - y_1)/N \tag{4}$$

$$\Delta z = (z_2 - z_1)/N \tag{5}$$

Therefore, the x, y, and z coordinate values of the interpolation point at any arbitrary (i+1) moment can be calculated as follows:

$$x_{i+1} = x_1 + i\Delta x \tag{6}$$

$$y_{i+1} = y_1 + i\Delta y \tag{7}$$

$$z_{i+1} = z_1 + i\Delta z \tag{8}$$

After obtaining the coordinate values of the interpolation points at any given moment, MoveIt will use the KDL solver to perform inverse kinematics calculations, deriving the angle for each joint. Subsequently, the robotic arm will move according to these calculated values. On the upper computer software, when Cartesian linear interpolation is selected and the joint point coordinates (0.5, 0.5, 0.5) are input, with RPY (Roll, Pitch, Yaw) set to 3.14, 0, and 0 respectively, and the execute button is clicked, the robotic arm will move to the target point. After reaching the target, if new coordinates (0.5, 0.1, 0.5) are input while maintaining the same orientation, the robotic arm will move along the planned linear trajectory. The motion pattern of the robotic arm is illustrated in **Figure 9**, with a total of 67 interpolation points and a movement duration of 6.7 seconds. As can be seen from the figure, the trajectory of the robotic arm consists of two straight lines, fulfilling the requirements of linear interpolation.

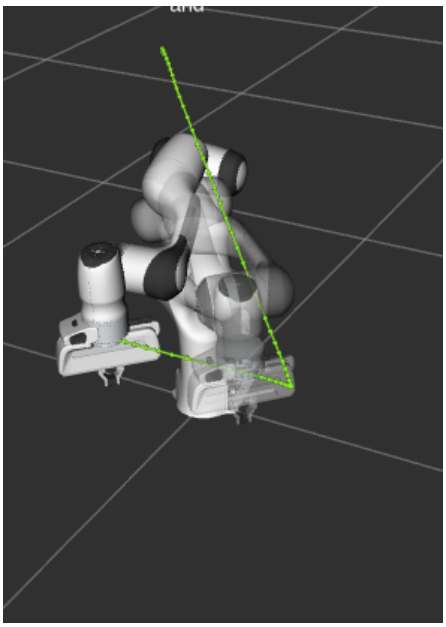

**Figure 9.** Linear interpolation of the robotic arm.

## 4.2. Circular interpolation

### 4.2.1. Principles of planar circular interpolation

A planar circular arc refers to an arc whose plane coincides with one of the three fundamental planes (e.g., the XOY plane). In this plane, given three non-collinear points A, B, and C, the circular arc formed by these points is illustrated in **Figure 10**.

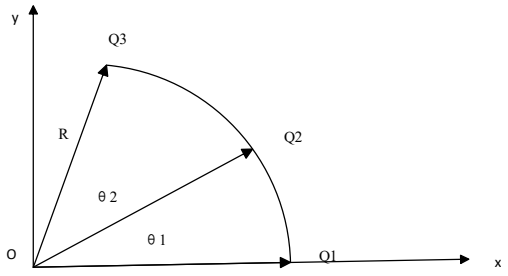


**Figure 10**. Planar circle defined by points Q1, Q2, and Q3.

R represents the radius of the planar circle, $\theta_1$ and $\theta_2$ are the central angles subtended by the arcs from Q1 to Q2 and from Q2 to Q3, respectively. Therefore, the total central angle is $\theta = \theta_1 + \theta_2$. Moreover, according to the chord length formula $L = 2R^* \sin\frac{\theta}{2}$, we have:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = 2R^* \sin\frac{\theta_1}{2} \tag{9}$$

$$\sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} = 2R^* \sin\frac{\theta_2}{2} \tag{10}$$

$$\theta_1 = 2arcsin\left(\frac{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}{2R}\right) \tag{11}$$

$$\theta_2 = 2arcsin\left(\frac{\sqrt{(x_3-x_2)^2+(y_3-y_2)^2}}{2R}\right) \tag{12}$$

Given the angular displacement $\Delta\theta = t_s v/R$ within a fixed time $t_s$ and the joint angular velocity $v$, the total number of interpolation steps can be calculated accordingly.

$$N = int(\theta/\Delta\theta + 1) \tag{13}$$

$\partial_i = i\Delta\theta$ represent the central angle from the starting point to the interpolation point at time instant i. Therefore, the coordinates of the interpolation point at time instant i+1 can be derived as follows:

$$y_{i+1} = R\sin(\partial_i + \Delta\theta) \tag{14}$$

$$x_{i+1} = R\cos(\partial_i + \Delta\theta) \tag{15}$$

### 4.2.2. Principles of spatial circular arc interpolation

Spatial circular arc interpolation is based on planar circular arc interpolation and can typically be carried out in three steps as follows:

(1) Transform the three-dimensional spatial problem into a two-dimensional planar problem by identifying the plane determined by three non-collinear points, which serves as the plane where the planar circular arc lies;

(2) Utilize the planar circular arc interpolation algorithm to calculate the coordinate values of each interpolation point within this plane;

(3) Compute the transformation matrix between the coordinate system established by the planar circular arc

and the base coordinate system, and then convert the coordinates of each interpolation point into their corresponding values in the base coordinate system.

As shown in **Figure 11**, a plane and a circular arc with a known radius R can be determined by three non-collinear points P1, P2, and P3. Through establishing a coordinate system named $O_RX_RY_RZ_R$ on this plane, making the origin of the coordinate system coincide with the center of the circle determined by points P1, P2, and P3, where $Z_R$ is the outward normal to the $O_RX_RY_R$ plane where the planar circular arc lies, planar circular arc interpolation theory can be applied to the circular arc defined by points P1, P2, and P3 in the plane.

If the rotation angle between the coordinate axis $Z_R$ of $O_RX_RY_RZ_R$ and the base frame is $\partial$, the angle between $XR$ and the X-axis is $\theta$, Given that $O_R$ in the original coordinate system are ($x_0$, $y_0$, $z_0$), the rotation matrix for rotating the $O_RX_RY_RZ_R$ by an angle  about the Z-axis is:

$$R_{zR}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

The rotation matrix for rotation by $\partial$ is:

$$R_{X_R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\partial & -\sin\partial \\ 0 & \sin\partial & \cos\partial \end{bmatrix} \tag{17}$$

So, the overall rotation matrix is:

$$R = R_{Z_R}R_{X_R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\partial & -\sin\partial \\ 0 & \sin\partial & \cos\partial \end{bmatrix} \tag{18}$$

Transformation matrix is as shown:

$$T = \begin{bmatrix} \cos\theta & -\sin\theta\cos\partial & \sin\theta\sin\partial & x_0 \\ \sin\theta & \cos\theta\cos\partial & -\cos\theta\sin\partial & y_0 \\ 0 & \sin\partial & \cos\partial & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

Therefore, the relationship for converting the coordinate values of the interpolation point P into the base coordinate values Q is as follows:

$$\varrho = TP \tag{20}$$

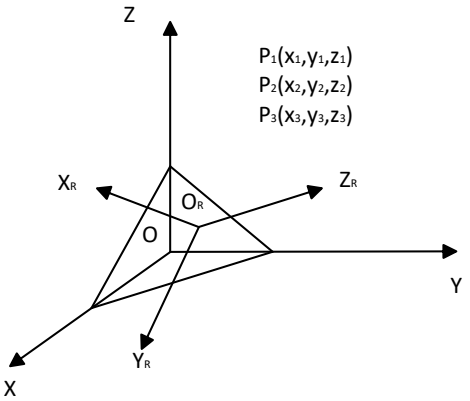


**Figure 11**. Base coordinate system and planar circular arc coordinate system.

On the upper-computer software, circular arc interpolation is selected. After inputting the coordinate values (0.5, 0.5, 0.5) and clicking the execute button, the robotic arm reaches the target point. Subsequently, another set of coordinates (0.5, 0.1, 0.5) is input, and the execute button is clicked again. The execution results, as shown in **Figure 12**, indicate that the number of interpolation points is 155, and the total time taken is 15.5 seconds. The interpolated trajectory consists of two arcs, meeting the requirements for circular arc interpolation.

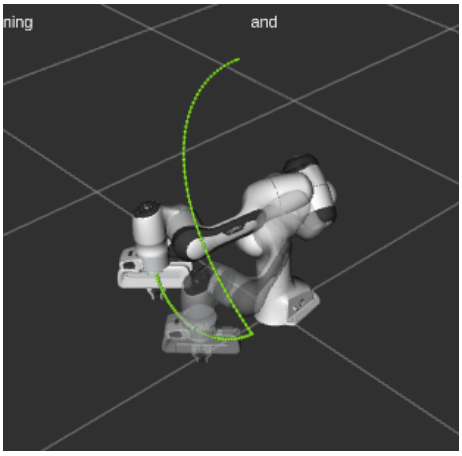


**Figure 12**. Arc trajectory interpolation.

Through simulation, it was found that when comparing the number of interpolation points and the time required for linear interpolation and circular arc interpolation under the same starting and ending coordinates, the number of interpolation points and the time for linear interpolation were significantly lower. However, during the experiments, it was also discovered that linear interpolation sometimes failed to reach the ending coordinates. Therefore, before connecting to the actual robotic arm, simulation can be utilized to observe in advance whether linear interpolation can reach the target point. If linear interpolation can reach the ending point, it can be given priority to reduce the number of interpolation points and calculation time. In other cases, circular arc interpolation should be used. During the operation of the robotic arm, the rqt tool can be opened, and the Visualization/Plot plugin under Plugins can be selected to observe the angular data of each joint of the robotic arm, as shown in **Figure 13**. It can be seen that after interpolation calculations are performed by MoveIt's internal algorithm, the angles of each joint of the robotic arm change smoothly.

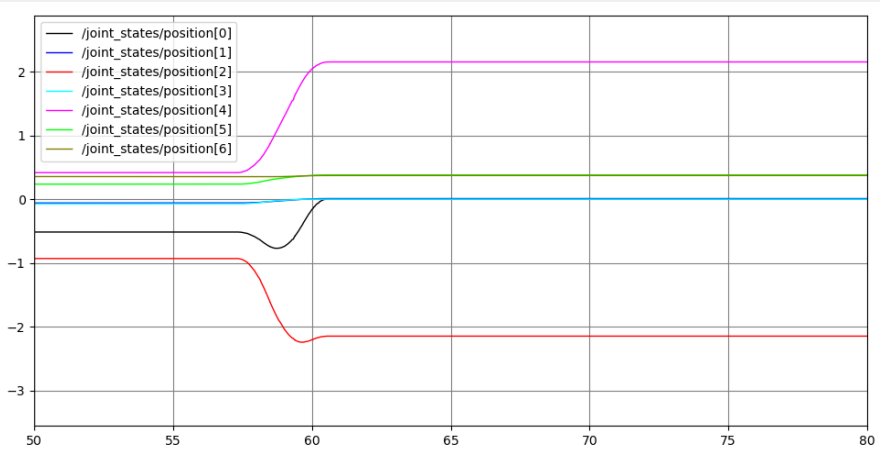


**Figure 13**. Joint angle curves.

## 5. Conclusion

This paper takes the Panda robotic arm as the research subject, displaying its status and controlling its motion through upper-computer software. By introducing obstacles into the planned scenarios, it compares the random sampling efficiency of the RRT algorithm and the RRT* algorithm in path planning for the robotic arm. Furthermore, it elaborates on the principles of linear and circular arc interpolation following path planning, as well as the importance of conducting simulations before connecting to the actual robotic arm. Through simulations, a smooth operation of each joint of the robotic arm was observed, laying a theoretical foundation for future research on robotic arms based on ROS2.

## Disclosure statement

The author declares no conflict of interest.

## References


[1] Cao H, Zhang X, Zhang Z, et al., 2022, Research Status and Trend Analysis of ROS-Based Real-Time Control Systems for Robotic Arms. Computer Measurement & Control, 30(3): 1–7.

[2] Hu J, 2022, Obstacle Avoidance Research for Six-Axis Industrial Robots Based on ROS. Science & Technology Information, 20(14): 10–12.

[3] Qu L, Gao K, Xing Y, et al., 2022, Research on Motion Planning for Robotic Arms based on ROS. Machine Tool & Hydraulics, 50(22): 43–47.

[4] Zhao P, Hong R, Fang C, 2022, Design of a UR5 Robot Control System based on ROS. Journal of Nanjing Tech University (Natural Science Edition), 44(2): 161–168.

[5] Wang B, 2023, Research on Motion Planning for Six-Axis Robotic Arms based on ROS. Computer Programming Skills & Maintenance, 2023(3): 140–142.

[6] Cheng L, Hua J, Hu J, 2022, Research on Trajectory Planning and Simulation for 6-DOF Robotic Arms based on ROS Platform. Manufacturing Automation, 44(1): 38–41.

[7] Hui D, Chen G, Ma J, et al., 2023, Modeling and Motion Planning for Robotic Arms based on ROS. Journal of Nanjing Institute of Technology (Natural Science Edition), 21(1): 52–58.

[8] Jiang H, Chao Y, Zhou J, et al., 2023, Improved RRT Algorithm for Robotic Arm Path Planning. Mechanical Design and Manufacturing, 2023(12): 288–292.

[9] Zhang B, 2022, Research on Motion Control of a 6-DOF Robotic Arm based on ROS, thesis, Jilin Institute of Chemical Technology.

[10] Zhang S, 2021, Trajectory Planning and Grasping for Robotic Arms based on ROS System, thesis, Dalian University of Technology.