ROKAE

xCore
Control System
Operation Manual

A Partner You Can Rely on in Production

Control System Version: V1.7
Version:

Contents in this manual are subject to change without notice. We assume no responsibility for any errors that may appear in this manual.

Please understand that in no event shall we be liable for incidental or consequential damages arising from the use of this manual and the products described herein.

We cannot foresee all possible dangers and consequences. Therefore, this manual cannot warn the user of all possible hazards.

No part of this manual may be reproduced in any form.

If you find the contents of this manual wrong or in need of improvement or supplement, please contact us for correction.

This manual is originally written in Simplified Chinese. Other language versions are translated.

Contents

1 File list

xCore Control System offers an array of basic functions and extended functions. The following files are available to help you master xCore quickly. Contact us if you need them.

ID	Name	Latest version	Introduction
1	*xCore Control System User Manual*	V1.6.1	Describes the basic functions of the xCore Control System;
2	*xVision User Manual*	V1.0.0	Describes the basic functions of xVision;
3	*User Manual for Laser Welding Process Kit*	V3.0	Describes the use of the Laser Welding Process Kit;
4	*User Manual for xCore Control System Extended Functions*		Describes the use of RCI and xCore-SDK;
5	*Operation Manual for Plating Line Tracking*	V1.2	Describes the use of Plating Line Tracking;
6	*RokaeStudio User Manual*	V1.1.0	Describes the use of off-line programming software;

2 Glossary

- HMI: Human Machine Interface;
- HMID: HMI device;
- RCI: Rokae Control Interface, external control interface for ROKAE robots, with real-time underlying control supported;
- SDK: Software Development Kit, which will gradually replace RCI to enable underlying robot control through C++ and other languages;
- Project: The collection of programs, tasks, and other objects that control the operation of the robot; data objects of a project can be exported and reused in other projects or robots;
- Task: In xCore, it is as it suggests;
- Module: It refers to a program file in xCore;
- Elbow: It is the angle between the arm plane and the reference plane. The arm plane refers to the plane formed by the robot's lower arm and upper arm, and the reference plane refers to the arm plane formed when the three axes are set to zero and the end-effector reaches the same pose;
- RL: Rokae Robot Language. It provides various commands to assist the robot in building projects;
- Robot Assist: It is the software launched by ROKAE that integrates the Teach Pendant function. Together with the new-generation control system xCore. It can be used for functions such as robot motion control, programming, parameter configuration, and status monitoring. It can run on xPad2 teach pendant, PC, and other devices;

3 Introduction

Overview

Robot Assist is the software launched by ROKAE that integrates the Teach Pendant function. Together with the new-generation control system xCore, it can be used for functions such as robot motion control, programming, parameter configuration, and status monitoring. Featured with a friendly interface, it supports all current ROKAE robots (including industrial robots and cobots) and will support newly-developed robots with ongoing updates. The software can be installed on PC, Surface, and ROKAE's Teach Pendant xPad2. The devices can control a robot after getting connected to it as long as they are in the same network segment as the robot.

Operating Environment

Besides xPad2, we suggest using a tablet or a laptop as the operating terminal. The recommended configurations are shown in the table below.

Terminal	Tablet	Terminal	Laptop
Storage Capacity	32GB	Storage Capacity	32GB
System Memory	4GB	System Memory	4GB
Screen Size	8.0 inches and above	GPU	Intel HD Graphics 4000 and above
Network Communication	Wi-Fi	Network Communication	Wi-Fi or Wired LAN
Operating System	Windows7 64bit, Windows10 64bit, Ubuntu16.04, Ubuntu18.04		
CPU	Intel Core I3 and above		

3.1 Main Interface

The main operation interface consists of the top status bar, the main display area, and the bottom status bar.
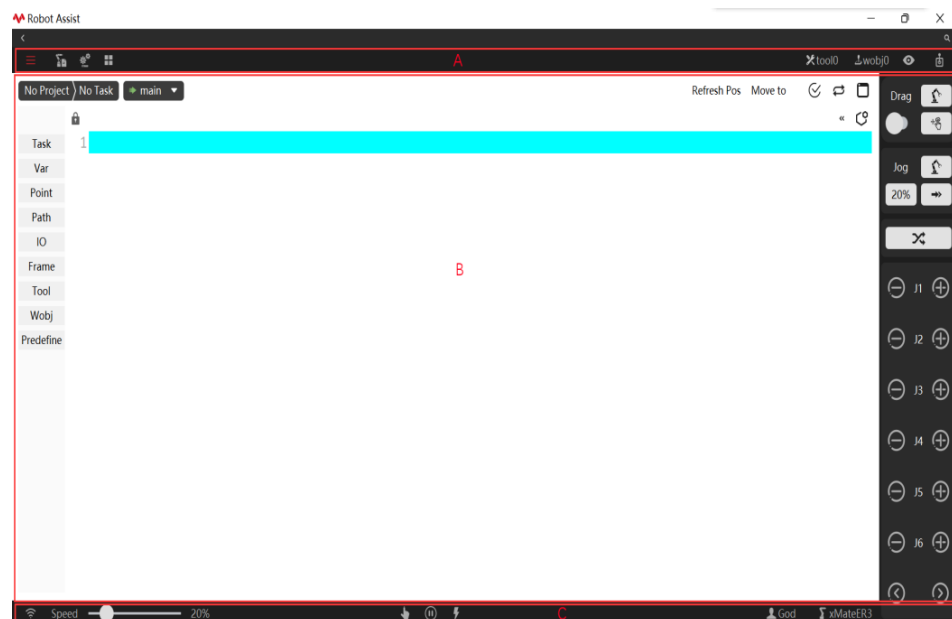
A	The top status bar shows the buttons for entering the main menu, robot programming, robot configuration, Teach Pendant option, Instant Log, Tool & Work Object Information, Status Monitoring interface, and operation interface.
B	The main display area shows the operation interface of each function module.
C	The bottom status bar shows the connection status, the program running rate, operating mode, motion

status, user information, and robot model.

Notes

Robot Assist interacts with the controller in real-time. Frequent changes in window size may cause the interface to stop refreshing. In this case, restore by pressing Alt+Tab to switch between windows.

3.1.1 Top Status Bar

Explanation

The top status bar shows the buttons for entering the Main Menu, Current Project, Instant Log, Tool & Work Object Information, Status Monitoring interface, and operation interface.

A	Menu button - Click to select the desired function module such as diagnostic log, Help, and demo and enter the corresponding sub-interface.
B	Robot Programming button - Click to enter the Current Project sub-interface.
C	Robot Configuration button - Click to enter function modules such as configuration, security, communication, and authorization.
D	Teach Pendant option button - Click to enter function modules such as connection, basic settings, appearance adjustment, and file manager.
E	Tools - Display the information of the tool currently in use and select the tool to be used.
F	Work Objects - Display the information of the work object currently in use and select the work object to be used.
G	Status Monitoring interface button - Click to open/close the status monitoring interface.
H	Operation interface button - Click to open/close the operation interface.

3.1.2 Bottom Status Bar

Explanation

The bottom status bar shows the connection status between Robot Assist and the robot, the program running rate, the robot operating mode, robot status, motor status, current user information, and robot model.

A	Connection status between Robot Assist and the robot - Red slash is for disconnected and full gray is for connected.
B	Program running rate adjustment slide - Adjust the motion speed when the program is running. Adjustable range: 1% - 100%. This parameter is valid for both manual and automatic mode program operation rates.
C	Robot operating mode - Click to toggle between Manual and Automatic.
D	Robot status - Includes the robot motion status, system status, controller mode, etc.
E	Robot motor status - Red is for powered-on and gray is for powered-off, other statuses include emergency stop and safety gate open. Click the button to power on the robot in Automatic mode.
F	Current user information: operator, admin, and god. Click the button to enter the user login interface. The default login password is 123456.
G	Robot model information.

Robot operating mode

 The Manual mode is used for robot programming and debugging.

In Manual mode, all robot motions are controlled manually by the user, and the robot will power on the motor and respond to the motion commands only when it's enabled (the three-position switch is in the middle position).

 The Automatic mode is used for continuous automated production,

in which the three-position enabling switch will be bypassed and the robot can work normally without manual intervention.

When the robot is in Automatic mode, the system IO signals can be used to control the robot or to obtain the robot's operating status. For example,

one DI signal can be used to start/stop the RL program and the other to control the motor power-on.

Robot status

The ⏸ button on the bottom status bar shows the robot motion status, system status, controller mode, etc.

Idle		The program is stopped and the robot is not in motion.
Program running		The program is running. The button turns red when the robot is in motion.
Drag mode		The controller can be dragged when it is in Drag mode. The button turns red when the robot is in motion.
Demo mode		The controller plays the Demo when it is in Demo mode. The button turns red when the robot is in motion.
Identification mode		The controller is in Identification mode. The button turns red when the robot is in motion.
Jog mode		The controller is in Jog mode and changes with the start and stop of the Jog button.
RCI mode		The controller is in RCI mode. The button turns red when the robot is in motion.
Collaboration mode		The controller is in Collaboration mode, which is displayed in combination with other status in the upper right corner of the icon.
Error		An error occurs in the robot system.
Debug mode		The controller is in Debug mode. The button turns red when the robot is in motion.

Robot motor status

Powered-on		The robot motor is powered on.
Powered-off		The robot motor is powered off.
Emergency stop	E	The robot is in the emergency stop state. The robot motor cannot be powered on.
Safety gate open	G	The safety gate is open. The robot motor cannot be powered on.

3.2 Status Monitoring

Explanation

Click 👁 on the top status bar to open the floating status monitoring interface, which monitors the robot 3D model, task running status, IO signals, network connection, and register variables, which are convenient for Jog and programming.

3.2.1 3D model monitoring

Explanation

The robot 3D Model interface displays the current 3D model of the robot, joint angles, joint torque, elbows, the position of the robot end-effector in a certain frame, and the RPY angle and quaternion of the robot end-effector relative to the rotation matrix of the base frame.

The robot 3D Model interface displays three frames: flange frame, base frame, and world frame. When the base frame is calibrated, the 3D model monitoring interface and the base frame relative to the world frame will change according to the calibration result.

The end-effector position can be displayed in three frames, namely the work object frame, the base frame, and the world frame; the monitor data in the base frame is displayed by default.
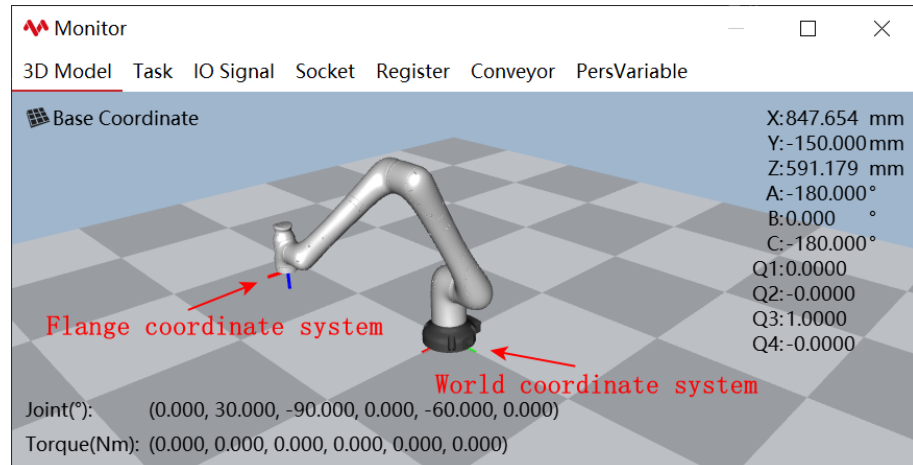
3.2.2 Multi-task monitoring

Explanation

The Multi-task monitoring interface displays the task type and running status of each task.

	Operation	Description
1	Task name	Tasks added to the project are displayed here.
2	Task type	The task type can be configured in Project -> Task List.
3	Start running the program to view the operation status of the task.	Stop task: STOP Run task: GO

3.2.3 IO signal monitoring

Explanation

For xMate cobots, the IO signal monitoring interface displays the 4-channel DI and DO signals on the robot base and the 2-channel DI and DO signals at the end-effector by default. For industrial robots, the IO signal monitoring interface displays configured IO signals in the control cabinet by default.
The IO simulation model can be turned on to test DI/DO signal value.
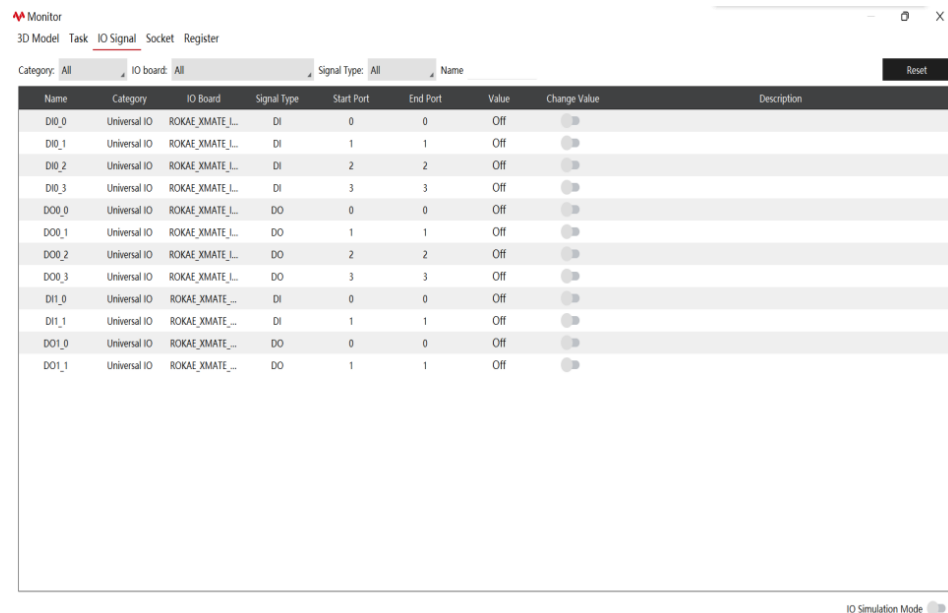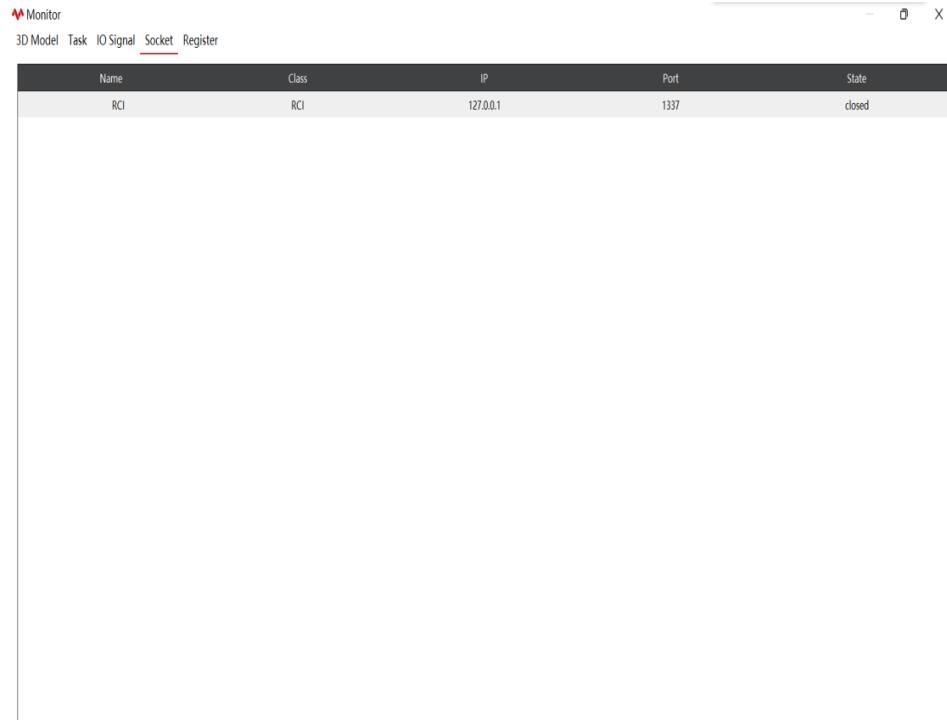
	Operation	Description
1	Go to the IO Signal page and click the [IO Simulation Mode] enabling switch to activate Simulation mode.	Only Admin or God users can activate this mode.
2	Click the DI/DO value buttons to start the simulation.	Note that even not in the Simulation mode, DO can also be forced to output.
3	Click the [IO Simulation Mode] button to turn off the Simulation mode.	The actual value and the modified value button are not strongly correlated, and the modified value button will be set to false after the simulation Mode is turned off.

3.2.4 Network connection monitoring

Explanation

The network connection monitoring interface displays the information (IP address, port number) and status of the network connections that are established with the controller. The connection status of SOCKET, MODBUS, and RCI are displayed by default.

	Operation	Description
1	[Type] The connection status of MODBUS, RCI, and SOCKET can be displayed.	The connection can be added and configured in the relevant interface. SYS_SOCKET refers specifically to the connection of external communication.
2	[Name] The name of the connection.	MODBUS, RCI, and SYS_SOCKET are system default unique names. User-defined names are displayed for new connections.
3	[IP] The IP address of the connection.	For a client-side connection, the IP address of the server is played. For a server-side connection, its own IP address is played.
4	[Port] The port number of the connection.	For a client-side connection, the port number of the server is played. For a server-side connection, its own port number is played.
5	[Status] The current status of the connection.	Generally, there are three types of connection status: Connected, Disconnected, and Connecting. For a server-side connection, it displays Monitoring when it is disconnected.

3.2.5 Register variable monitoring

Explanation

The register variable monitoring interface displays the information for each register. The content filter is available for quick search.
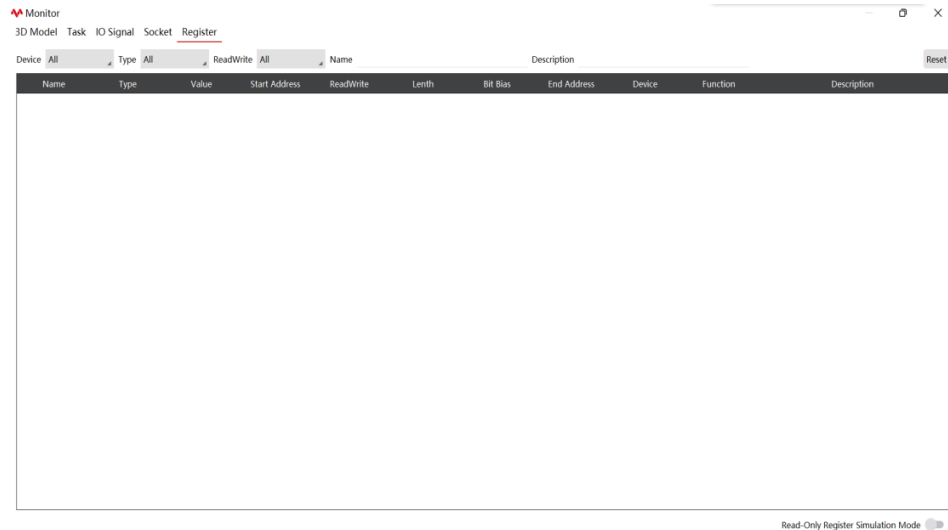
	Operation	Description
1	The user can customize content to be displayed using [Content Filter].	The variable can be filtered by connection, variable type, name, description, and others for a clear view.
2	Please refer to the Modbus register variable configuration for the definition of each column.	

3.3 Operation interface

Explanation

Click or on the top status bar to open the operation interface, which can be used to change the robot control mode, control robot motion, and perform pose teaching.

The robot supports two types of pose teaching: JOG Mode and Drag Mode (for xMate cobots only).

➢ In Jog Mode, the Jog button is used to control the motion in the corresponding directions.

➢ In Drag Mode, directly and manually guide the robot's motion using the end-effector drag Pilot handle or xPanel.

No.	Description
A	Drag settings zone, which shows that B, C, and D are drag-related options.
B	Drag space setting: joint space drag and Cartesian space drag.
C	Drag enabling switch - turn on/off Drag Mode.
D	Drag mode setting. For joint space drag, only free drag is available. For Cartesian space drag, the three options of translational drag only, rotational drag only, and free drag are available.
E	JOG setting zone. It shows that F, G, and H are jog-related options.
F	Jog reference frame setting - Select single-axis mode or Cartesian mode in Jog Mode and select the reference frame in Cartesian mode, including world frame, base frame, flange frame, tool frame, and work object frame.
G	Jog speed setting - Set the robot Jog speed between 1% and 100% (expressed in a percentage relative to the top Jog speed limit of 250 mm/s).
H	Jog step mode setting - Set Jog mode to Continuous Jog or Stepping Jog, and the stepping increment can be adjusted.
I, K	Switch function area - Switch between the Jog button and buttons L-Q.
J	Jog button. For a 7-axis robot, J1 to J7 are displayed in the case of joint space Jog and X/Y/Z/A/B/C and Elbow in the case of Cartesian space Jog. For a 6-axis robot, J1 to J6 are displayed in the case of joint space Jog and X/Y/Z/A/B/C in the case of Cartesian space Jog.
L	Program start/stop button.
M	Program running buttons - previous/next.
N	Move to Zero Pose button.
O	Move to Drag Pose button.
P	Move to Shipping Pose button.
Q	Move to Customized Home Pose button.
R	"Screenshot" button, to take a screenshot and save it to the local folder of the teach pendant. These screenshots can be exported on the "Basic Settings" interface. The button is only shown on the teach pendant. Note: When taking a screenshot on the teach pendant, the physical membrane button is recommended. In the case of a pop-up window, the "Screenshot" button becomes inactive, and clicking the button generates no response.

ℹ Notes

Please ensure the robot is currently in 🖐 manual mode and ⚡ powered off before performing Jog and turning on the drag enabling switch.

3.4 Function module
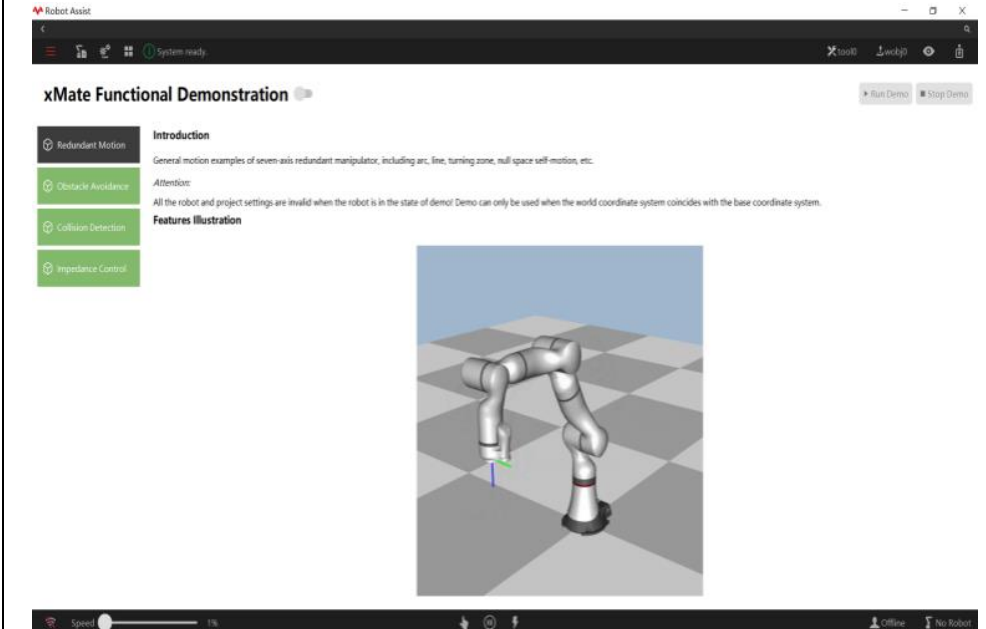
Explanation

Click the menu button ☰ to open the function tabs. The function modules include the engineering module, robot module, diagnosis module, demo module, option module, and help module. Click the main menu button to switch between different function module tabs.

3.4.1 Menu module

Diagnostic log	
Teach pendant log	It displays the log of the Robot Assist (HMI software);
Controller logs	It displays the running log of the controller connected to the robot;
Log timeline	It displays the log history visually through a timeline;
Internal logs	It displays the underlying log information of the controller. In case of a robot failure, Technical Support can quickly locate the cause of the problem by viewing the internal logs;
Advanced options	They are used to assist developers with the diagnosis of the servo, ECAT, and other equipment, and enable real-time thread alarm and monitoring and more. Since enabling the diagnostic function will increase the workload of the controller, do not turn it on in actual production unless necessary.

Help	
About ROKAE	A brief introduction to the interface, controller, and ROKAE robots
Introduction to Products	It contains all specifications of industrial robots and cobots
Software Upgrade	It provides functions such as controller upgrade, controller backup, restoring factory settings, restarting the robot, and erasing the robot configuration information

Demos
Four demos are provided to demonstrate xMate7 features such as DOF design, agility of redundant motion control, one-touch stop sensitivity, and compliance of variable impedance stiffness.

3.4.2 Robot programming module

Interface functions
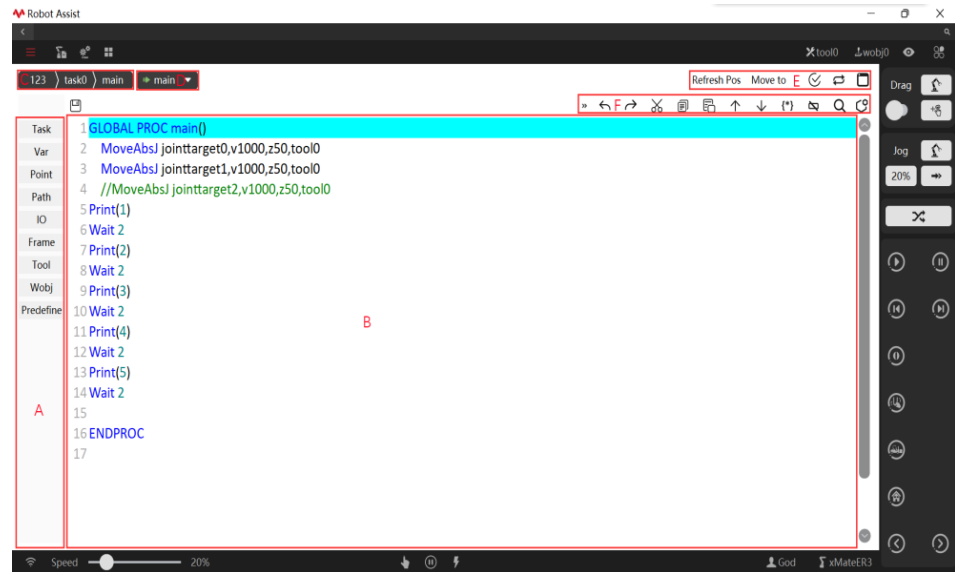
A	Tabs - used to switch between the Project Sub-objects Setup interfaces, including Task, Variable, Point Position, Path, IO Signal, User Frame, Tool Frame, Work Object Frame, Predefinition, etc.;
B	Program edit area - for auxiliary programming of the robot and program command;
C	Program file selection - used to switch between different tasks and program files for editing and debugging;
D	Program Debug Quick Positioning button - used to switch to main function or cursor;
E	Program syntax check, loop mode, and output terminal;
F	Program edit toolbar: undo, repeat, cut, paste, copy, move up one row, move down one row, batch comment, delete current row, search and replace, auxiliary programming;

3.4.3 Robot configuration module

Set	User group, controller settings, zero-point calibration, base frame calibration, dynamic parameter identification, robot body parameters, kinematics parameters, force control parameters, quick turn;
Safety	Soft limits, virtual wall, collision detection, safety area, safety monitor, collaboration mode;
Communication	System IO, external communication, register, IO device, bus device, end-effector tool, RCI settings, serial port settings;
Process kit	Laser welding, etc.;
Authorization	EtherCAT authorization;

3.4.4 Teach Pendant option module

Connection	It includes robot detection, robot connection, and auto reconnection settings;
Basic settings	It includes software language settings, auto startup settings, IP binding, working area path selection, graphic performance adjustment, and turning off 3D display;
Appearance Adjustment	It includes theme switching, control adjustment, and font adjustment.
File Manager	It includes opening and browsing of cache folder, log folder, working area folder, etc.

4 Connecting to the Robot

Explanation
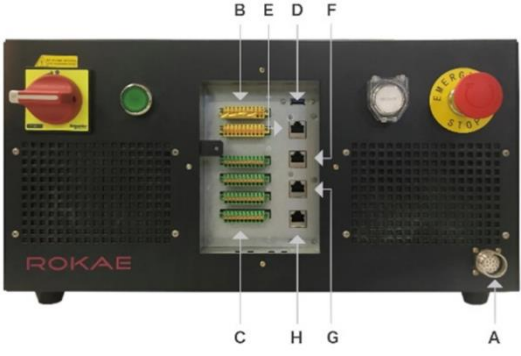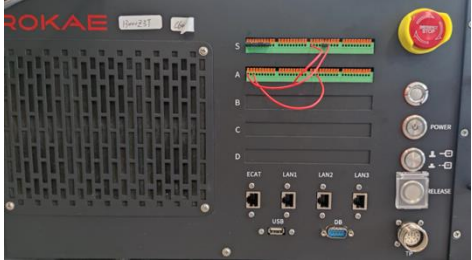
The devices (xPad2 teach pendant, PC) on which Robot Assist is running can be connected to any Rokae robot as long as they are in the same LAN as the robot. The connection with the robot can be established by robot detection or by manually entering the controller address.

The robot system only supports wired connection to the local area network (LAN) or direct connection with a network cable:
> When using the Teach Pendant xPad2, directly connect xPad2 to the corresponding interface of

the robot;

➢ When using a PC on which Robot Assist is running to debug a robot, the PC can be directly connected to the robot via network cable;

➢ When switching between multiple robots, the robots can be connected to the same LAN and the PC on which Robot Assist is running will detect the robots available for connection on the same network segment;

➢ For scenarios where a wired connection is not convenient (such as on AGVs), the robot can be connected to a wireless router via the reserved network interface on the robot control cabinet (the network interface for xMate cobot base; the vision and debugging network interface of industrial robot control cabinet) and then to the HMID wirelessly.

4.1 Robot network interface and IP

ID	Introduction	Picture
1	The **xMate ER series** cobot has two Ethernet network interfaces on the base, of which the network interface J2 defaults to the fixed IP address of 192.168.0.160, and the network interface J1 defaults to an automatically obtained IP address.	
2	The **xMate CR series** cobot has only one Ethernet network interface J1 (standard configuration) on the base, which defaults to the fixed IP address of 192.168.0.160.	
3	For an industrial robot using the **XBC3 controller**, the controller features four longitudinally-arranged Ethernet network interfaces, they are, from top to bottom, EtherCAT device extension interface (used for slave station extension); vision network interface (used to connect industrial cameras, defaults to the fixed IP address of 192.168.2.160); debugging network interface (defaults to the fixed IP address of 192.168.0.160), and bus extension network interface (optional).	
4	For an industrial robot using the **XBC5 controller**, the controller features four horizontally-arranged Ethernet network interfaces, they are, from left to right, EtherCAT device extension interface (used for slave station extension); vision network interface (used to connect industrial cameras, defaults to the fixed IP address of 192.168.2.160); debugging network interface (defaults to the fixed IP address of 192.168.0.160), and bus extension network interface (optional).	

| 5 | | For the **XBC5 controller cabinet with high protection rating**, the controller cabinet reserves no network interface on the control cabinet. | |

4.2 Connecting to terminal devices

ID	Terminal devices	Description
1	xPad2	The Teach Pendant xPad2 supports the xMate CR series cobots and industrial robots using the XBC5 controller cabinet. Connect the Teach Pendant xPad2 to the robot via a cable plug (for industrial robots, the interface is on the XBC5 controller cabinet; for xMate cobots, the interface is on the base of the robots), and the connection is established. After the robot is powered on, the Teach Pendant xPad2 will be powered on automatically, and the pre-installed Robot Assist software will run automatically.
2	PC	**Method 1: Direct cable connection** Both the robot base and the controller cabinet feature one network interface that defaults as the debugging network interface and is assigned with the fixed IP address of 192.168.0.160. This IP address is the same for all robots and should not be modified arbitrarily. The PC on which Robot Assist is running can be connected to the network interface directly via a network cable to control the robot. **Method 2: External network interface connection** External network interface connection supports two types of settings: obtain an IP address automatically or assign a static IP address. Obtain an IP address automatically - Set the network interface J1 of cobots or the vision network interface of industrial robots to DHCP mode. The robot is connected to a router with DHCP via the network interface, which automatically assigns an IP address to the robot. The robot can then be detected and connected via robot detection. Assign a static IP address - Set the network interface J1 of cobots or the vision network interface of industrial robots to the IP address in the required network segment. The robot is connected to a router via the network interface and can then be visited and controlled via the robot's IP address. IP address modification: Using the Windows 10 operating system as an example, connect one end of the Ethernet cable to the robot's J2 interface and the other end to the terminal device (PC). Click on the "Start > Control Panel" menu on the terminal device (PC), and select "Network and Sharing Center". The "Network and Sharing Center" window will pop up. Click on "Local Area Connection" in the "Network and Sharing Center" window, and the "Local Area Connection Status" interface will appear. Click on "Properties" in the "Local Area Connection Status" interface, and the "Local Area Connection Properties" interface will appear. Double-click on "Internet Protocol Version 4 (TCP/IPv4)" in the "Local Area Connection Properties" interface, and the "Internet Protocol Version 4 (TCP/IPv4) Properties" interface will appear. Select "Use the following IP address" in the "Internet Protocol Version 4 (TCP/IPv4) Properties" interface, modify the IP address, subnet mask, and default gateway of the terminal device (PC), and confirm the changes. (The terminal device (PC) shares the same subnet mask and default gateway with the robot and may use any unoccupied IP address in the same network segment.)
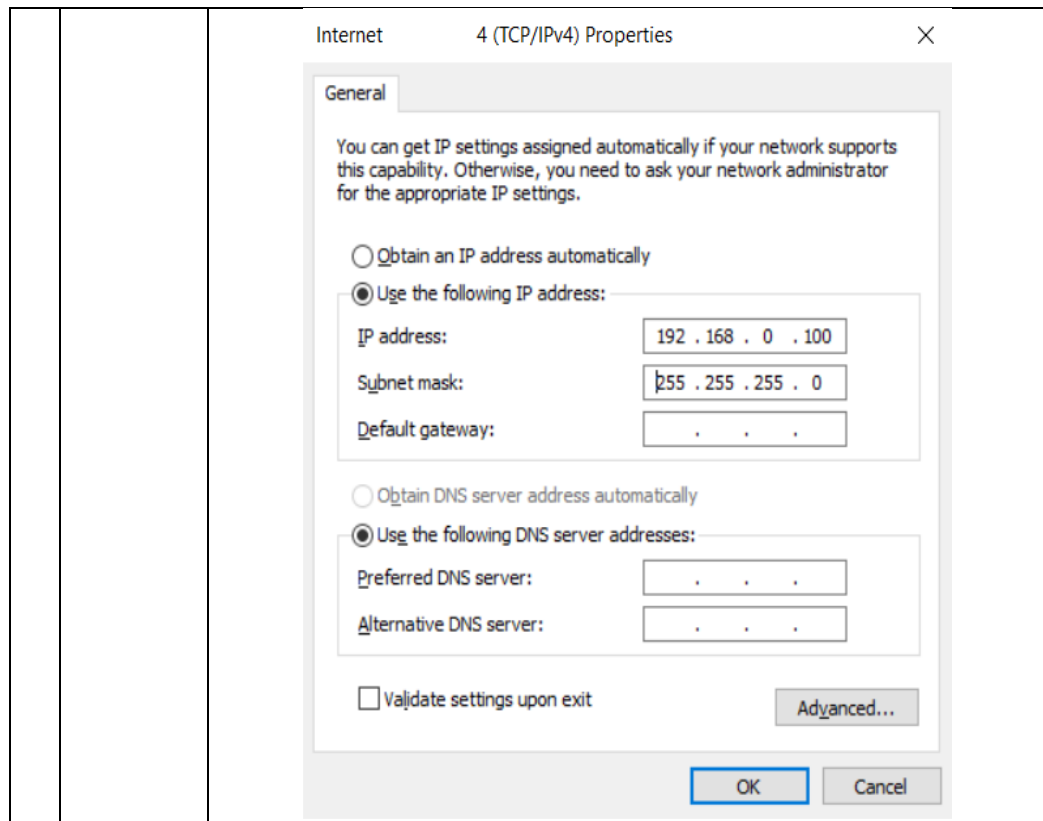
> ⚠️ Warning
>
> When manually modifying the IP address of the robot's network interfaces, do not set different network interfaces as static IP addresses of the same network segment; do not arbitrarily modify the network mode and IP address (192.168.0.160) of the debugging network interface; do not arbitrarily modify the network mode and IP address (192.168.1.160) of the Teach Pendant xPad's network adapter card.

4.3 Connecting to the robot

Connecting to the robot

Go to Options -> Connection interface and enter the IP address of the robots.
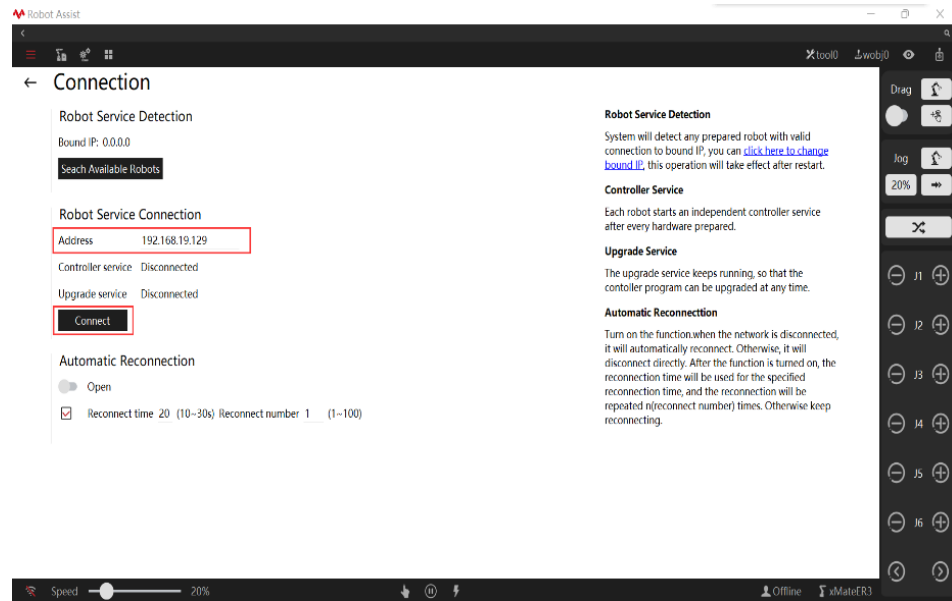
Notes

IP and ports are necessary for identifying the target controller. When you fail to connect to the robot, check to see if the network is connected.

Robot detection

Explanation

HMI can detect and display all robots available on the same network segment for connection.

Click the network icon button on the bottom status bar to enter the robot search interface, and click Search Available Robot.
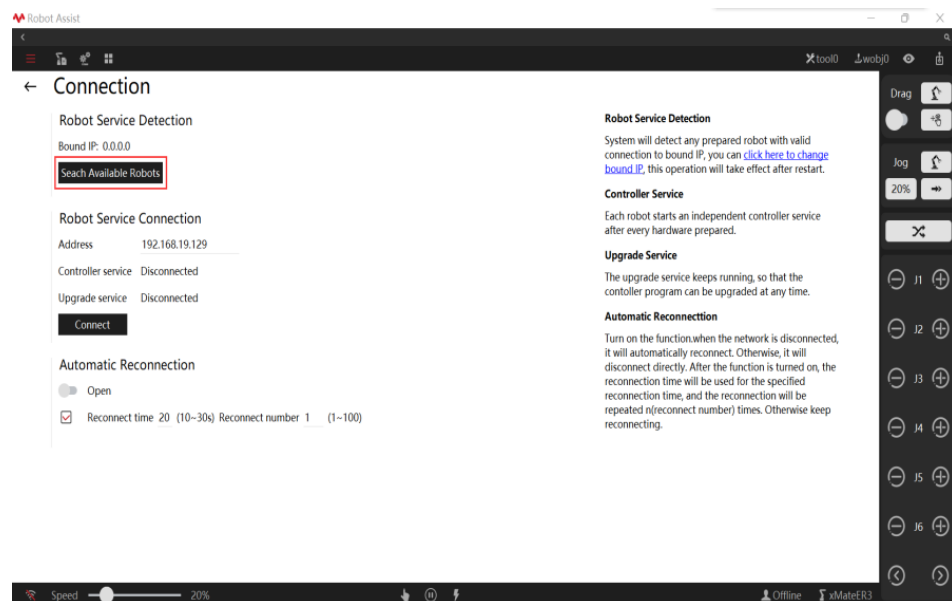
Notes

1.	When searching for robots, please make sure the device on which Robot Assist is running and the robots are on the same network and the network is connected.
2.	If the robots cannot be detected by searching for available robots when the device on which Robot Assist is running and the robots are on the same network and the network is connected, the connection requests sent by Robot Assist may be blocked by the firewall on the device.

4.4 User login

Explanation

The default user is Operator after the robot is successfully connected. Click on the bottom status bar to switch between users. The default password is 123456.

For details on user login and permissions, please refer to 5.1.1.1 User groups and permissions.

4.5 Disconnect and restore connection

Explanation

➢ Click the Disconnect button in the Connection interface to disconnect Robot Assist from the controller.

➢ Simultaneous connection of multiple Robot Assist is not supported. Another Robot Assist can only be connected after the current Robot Assist is confirmed to be disconnected or the robot is restarted.

➢ The Robot Assist connection can be restored via user login in the same way it is connected for the first time.

4.5.1 Auto reconnect

Explanation

In the Connection interface, the Automatic Reconnection function can be enabled by turning on the switch. There are two ways for reconnecting:

➢ Check the checkbox to set reconnection interval and attempts to specify reconnection interval and attempts (the total reconnection duration = reconnection interval * reconnection attempts).

➢ If the checkbox is not checked, no reconnection interval or attempts will be set and Robot Assist will keep reconnecting to the controller.
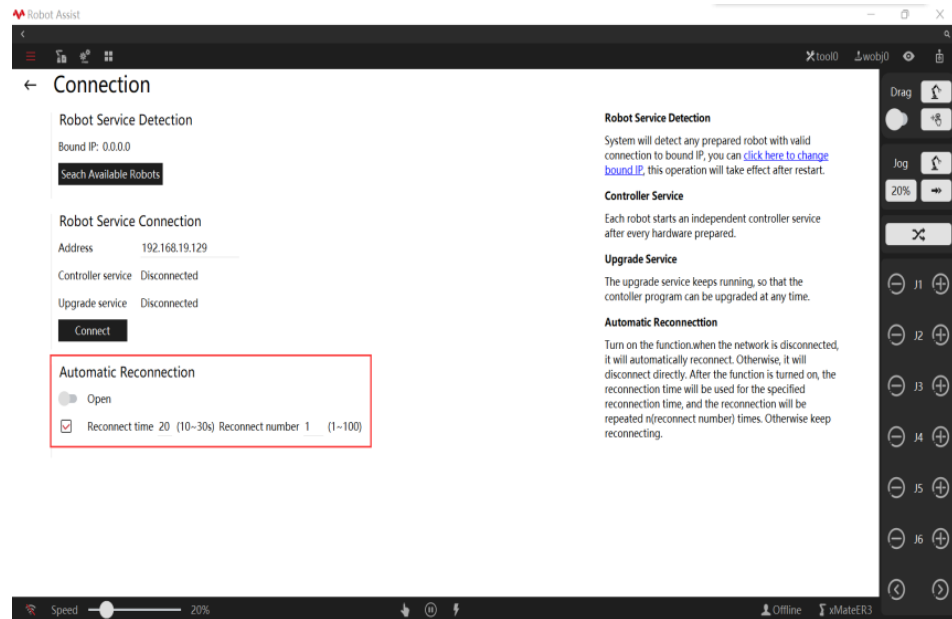
4.5.2 Plug & play Teach Pendant xPad2

Explanation

Follow the steps below to disconnect the Teach Pendant xPad2 from a powered-on robot. Direct physical disconnection will cause an emergency stop for the robot.

➢ Go to the Basic Settings interface, and the current status is displayed in Teach Pendant Mode Settings, which defaults to Teach Pendant Mode. Click the Switch Mode button to switch to No Teach Pendant Mode. No Teach Pendant Mode will be displayed on the interface after a successful switch.

➢ After the robot is switched to No Teach Pendant Mode, the Teach Pendant xPad2 can be disconnected from the robot, and the robot will not come to an emergency stop in this case.

To reconnect the Teach Pendant xPad2 to the robot, follow the steps below:

➢ Establish a physical connection between the Teach Pendant xPad2 and the robot;

➢ Go to the Basic Settings interface, and the current status is displayed in Teach Pendant Mode Settings, which should be No Teach Pendant Mode. Click the Switch Mode button to switch to Teach Pendant Mode. Teach Pendant Mode will be displayed on the interface after a successful switch.

> **ℹ Notes**
>
> The plug & play Teach Pendant function is only available for some models. For models that do not support this function, the system will prompt "Failed to switch Teach Pendant Mode". For detailed model configurations, please consult our technical support.

5 Operating Mode and Safety

5.1 Safety Management

5.1.1 About this section

This section introduces the safety principles and processes that need to be noted when using robots. The contents related to the design and installation of the external safety protection device of the robot are not within this section.

5.1.2 Safety terms

5.1.2.1 Safety symbols

About safety symbols

There may be different degrees of danger when operating the robot in accordance with this manual, so there will be a special safety symbol in the vicinity of dangerous operation instructions to remind the user to be careful. The contents include:

➢ An icon that indicates safety level and the corresponding name, such as warning, danger, prompt, etc.;

➢ A brief description given to illustrate the possible consequences if the operator does not eliminate the danger;

➢ The operating instructions on how to eliminate dangers.

Safety levels

Icon	Name	Description
	DANGER	For the contents that come with this sign, failure of following the rules in operation will cause serious or even fatal injury to personnel, and will/may cause serious damage to the robot.
	Warning	For the contents that come with this sign, failure of following the rules in operation may cause serious and even fatal personal injury and will cause great damage to the robot.
	Electric shock hazard	It indicates that the current operation may cause an electric shock hazard with a serious or even fatal injury.
	Caution	For those coming with this sign, failure of following the rules in operation may cause personal injury, and may cause damage to the robot.
	ESD	It indicates that the components involved in the current operation are sensitive to static electricity. Failure to operate according to specifications may cause damage.
	Notes	It is used to prompt some important information or prerequisites.

Hazard description

Icon	Name	Description
	Squeezing	There may be an injury to the operators and maintenance personnel who enter into the motion range of the robot during debugging, repair, overhaul, and tools clamping.

	Hands Pinching	The maintainers have the risk of hand pinching when approaching tape drive parts during the maintenance.
	Strike	There may be a serious injury to the operators and maintenance personnel who enter into the motion range of the robot during debugging, repair, overhaul, and tools clamping.
	Friction	There may be an injury to the operators and maintenance personnel who enter into the motion range of the robot during debugging, repair, overhaul, and tools clamping.
	Parts fly out	There may be a serious injury to the operators and maintenance personnel who enter into the motion range of the robot during debugging, repair, overhaul, and tools clamping when tools or work objects may fly out due to loose clamping.
	Fire	Electrical short circuits, burning wires/devices may cause fire hazards, causing serious injuries.
	Hot surface	During the maintenance and repair of the equipment, a burn may be caused if the maintenance personnel touch the robot's hot surface.

5.1.2.2 Safety features

Safety levels

The xCore system is equipped with specialized safety modules for handling safety-related signals, and provides external safety symbol interfaces such as safety gates and safety gratings.

The signals handled by safety modules include:

➢ Emergency stop signal;

➢ Safety gate signal;

➢ Enabling switch signal;

5.1.2.3 Stop

Type of stop

Rokae robot supports two types of stop: emergency stop and controlled stop.

Emergency stop	Emergency stop possesses the highest priority in the robot system. The emergency stop button, when pressed, will immediately trigger the emergency stop sequence. All other control functions as well as robot movements will stop, and the motor power for each joint will be cut off. The control system will switch over to the emergency stop state which will be maintained until a manual reset. After triggering the emergency stop, the system may take any of two different stop modes according to different working conditions: ➢ STOP 0: When the power is cut off, the robot stops working immediately. This is an uncontrolled stop. As each joint will stop as quickly as possible, the robot may deviate from the set path. Such a protective stop can only be used when the safety assessment limits are exceeded or there is an error in the safety assessment module of the control system; ➢ STOP 1: When the power supply causes the robot to stop, the power is cut off when the robot comes to a stop. This is a controlled stop and the robot will follow the set path. The power is cut off after the robot stops moving; **ℹ Notes** 1. Emergency stop is only used to stop the robot immediately in case of danger. 2. Emergency stop should not be used for normal stops, otherwise, it may cause extra and unnecessary wear to the brake and transmission system, which will eventually reduce the robot's service life.

Controlled stop	A controlled stop is to stop the program from running when the robot power is kept on. ➢ STOP 2: Controlled stop when the robot is powered on. The safety assessment control system can keep the robot at the stopped location.

5.1.2.4 Enabling switch

Enabling device

The enabling device is a special switch with two contacts and three positions and is also called a three-position enabling switch (hereinafter referred to as "enabling switch"). It is used to power on/off the robot power supply in Manual Mode to enable robot motions.

The motor power is switched on only when the enabling switch is pressed and kept in the middle so that the robot is in a state that is permitted for motion. Releasing or pressing the switch all the way down will cut the power off.
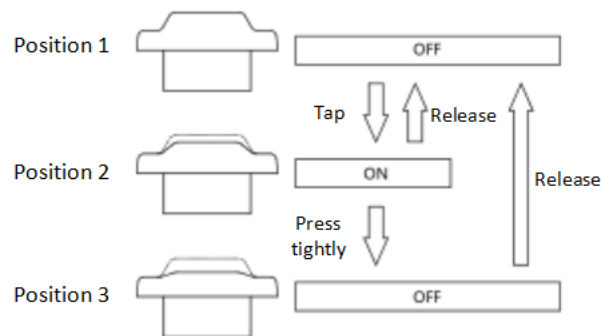
> **Notes**
>
> The yellow button on the Handheld Enabling Device is the enabling switch. When the enabling switch is pressed and held in the middle position, the robot will be powered on, the system will enter the Motor On state, and you can jog the robot or execute a program. The robot will be powered off and the system will return to the Motor Off state when the switch is released or pressed all the way down.
>
> In order to use the Teach Pendant safely, the following requirements must be observed:
> 1. Make sure the enabling switch functions properly in any circumstances.
> 2. Release the enabling switch immediately when no robot motion is required during programming or debugging.
> 3. Any person who enters the robot's working space must carry a handheld enabling device to prevent others from starting the robot without the knowledge of the involved personnel.

> ⚠️ **Warning**
>
> It is strictly prohibited to use external devices to keep the enabling switch locked or stopped in the middle position!

5.1.3 Safety precautions

5.1.3.1 Overview

About the robot

In human-machine collaboration, xCore offers safety functions such as Collaboration Mode and collision detection to ensure personal safety when collaborating with a robot. Please carefully read the safety functions in Section 7.2 before operating the robot.

About this section

This section will describe some basic safety specifications for the end-users of the robot. However, it cannot cover each specific circumstance due to limited space.

5.1.3.2 Focus on user's own safety

General principles

A few simple principles should be followed in order to operate the robot safely:

➢ Pay attention to the moving tools installed on the robot, such as the electric drill and electric saw. They shall be stopped when approaching the robot;

➢ Pay attention to the work object surface or the robot arm body. The motor and casing temperature of the robot may become very high after prolonged work;

➢ Watch out for grippers and objects gripped. If the gripper is opened, the workpiece could fall and cause personal injury or equipment damage. Moreover, the gripper of the robot may be very powerful and may cause injury if it is not used according to the specification.

5.1.3.3 Recovering from emergency stops

Explanation

In the case of an emergency stop, a reset is required to return to normal operation. The reset is a simple but important procedure. It ensures that the robot system is not returned to production in a hazardous condition.

Reset emergency stop button

All button-shaped emergency stop devices are equipped with one safety lock mechanism, which must be released manually after being pressed to reset the emergency stop status of the device.

Most emergency stop buttons are released by rotation and the direction of rotation is indicated on the button surface. Some buttons also support releasing by upward pulling.

5.1.3.4 Safety precautions in Manual mode

About the Manual mode

In Manual mode, the robot's movement is under manual control. You can jog the robot or execute a program only when the enabling switch is held in the middle position.

The Manual mode is used during programming, debugging, and commissioning of the workstation.

Speed limit in Manual mode

In Manual mode, the speed of the robot's end effector is limited to 250 mm/s. This means that the maximum speed of the robot will not exceed 250 mm/s whether you jog the robot or execute a program, regardless of the speed set in the program.

Bypassing external safety signals

In Manual mode, signals of external safety devices such as the safety gate and safety grating will be bypassed. This means that the emergency stop will not be triggered in Manual mode even if the safety gate is open, which facilitates the debugging.

5.1.3.5 Safety precautions in Automatic mode

About the Automatic mode

The Automatic mode is used for running the robot program in production.

In Automatic mode, the enabling switch will be bypassed so that the robot can run automatically without manual intervention.

Enabling external safety signals

External safety devices such as the safety gate and safety grating will be enabled in Automatic mode. Opening the safety gate will trigger an emergency stop.

Safe troubleshooting in production

In most cases, the robot is part of the production line. Therefore, the impact of a robot fault may go beyond the workstation itself. Likewise, problems with other parts of the production line may also impact the workstation. For this reason, a troubleshooting plan should be designed by personnel who are familiar with the entire production line to improve safety.

For example, a robot on the production line grabs workpieces from the conveyor belt. When the robot encounters a fault, the robot maintenance personnel should consider additional safety measures for working beside the moving conveyor belt to ensure uninterrupted production while the robot is under repair.

For another example, when removing a welding robot from the production line for routine maintenance, the robot supplying materials to it must also be stopped to avoid personal injury.

5.1.3.6 Emergency handling

5.1.3.6.1 Fire

Treatment of mild fire disaster

Do not panic and keep calm when a fire hazard is imminent or has not yet begun to spread; you can use on-site fire-extinguishing devices to put out the flame. It is strictly prohibited to use water to put out a fire caused by short circuits.

> ⚠️ Warning
>
> The fire-extinguishing device on the working field of the robot shall be supplied by the user, the user shall choose the appropriate fire-extinguishing device according to the actual situations of the field.

Treatment of severe fire disaster

If the fire has spread and is beyond control, the workers on site shall notify other workers immediately to give up their personal belongings and evacuate immediately through emergency exits rather than try to put out the fire. DO NOT use an elevator, and be sure to inform the fire department during evacuation.

If a person's clothing catches fire, ask them not to run but to lie flat on the ground immediately. Put out the fire using clothes or other suitable items and methods.

5.1.3.6.2 Treatment of an electric shock

Cut off power

When someone gets an electric shock, do not panic and cut off the power supply immediately. Appropriate methods and measures shall be adopted without hesitation according to specific site conditions. Generally, there are several methods and measures:

1) If the power switch or button is very near to the location of the electric shock, it shall be switched off at once, and the power supply shall be cut off.

2) If the power switch or button is far away from the location of the electric shock, it is suggested to use insulated pliers or ax, knife, and shovel with dry wooden handles to cut off live wires on the mains' side (power supply), the separated wire must not contact with the human body.

3) If the conducting wire is over or under the body of the victim, it is suggested to use a dry stick, board, bamboo pole, or other tools with insulated handles (by gripping the insulated handle) to remove the wire. No metal bar or wet object shall be used to avoid the rescuer from also getting an electric shock.

Treatment of the wounded after being separated from the power source

1) If the wounded is conscious, he/she shall be made lie on the back and watched out. He/she is not suggested to stand or walk for the time being.

2) If the wounded is unconscious, make him/her lie on the back to keep the airway open. Call the wounded or pat him/her on the shoulder at an interval of 5 seconds to judge if he/she loses consciousness. Do not call the wounded by shaking his/her head. Meanwhile, contact the hospital as soon as possible.

3) If the wounded loses consciousness, his/her respiratory conditions and heartbeat shall be confirmed within 10 seconds. If neither breath nor arterial pulse is sensed, the wounded may have a cardiac arrest and shall be given immediate first aid treatment by cardiopulmonary resuscitation.

5.2 Robot operating mode

5.2.1 Manual mode

Explanation

The Manual mode is mainly used for robot programming and debugging.

In Manual mode, all robot motions are controlled manually by the user, and the robot will power on the motor and respond to the motion commands only when its motion is enabled (the three-position switch is in the middle position).

Tasks typically performed in Manual mode

The Manual mode is typically used to perform the following tasks:

➢ Jog the robot back close to the path after an emergency stop to continue running the program;

➢ Create and write RL programs;

➢ Debug the RL program, including but not limited to startup, stop, single-step run, and teaching point update;

➢ Set control system parameters and calibrate frames;

➢ View and modify variables;

5.2.2 Automatic mode

Explanation

The Automatic mode is used for continuous automated production, in which the three-position enabling switch will be bypassed and the robot can work normally without manual intervention.

When the robot is in Automatic mode, signals can be used to control the robot or to obtain the robot's operating status. For example, one DI signal can be used to start/stop the RL program and the other to control the motor power-on. See the Section 7.3.1 for the list of system IOs supported by the xCore system.

Tasks forbade in Automatic mode

The Automatic mode is typically used to perform the following tasks:

➢ Load, start, and stop the RL program;
➢ Return to the original programming path after an emergency stop;
➢ Back up the system;
➢ Clean the tools (according to the process requirements);
➢ Machine and process the work objects;

Restrictions in Automatic mode

➢ Jog the robot.
➢ Modify configuration files, configure the number of IO boards, or set the robot installation method.
➢ Restore the backup.
➢ Grant function authorization.
➢ Set soft limits.
➢ Create, modify, and delete IO.
➢ Perform parameter identification.
➢ Turn on/off collision detection.
➢ Turn on/off Collaboration Mode.
➢ Turn on/off Drag Teaching in Automatic Mode.
➢ Perform calibration.
➢ Create new variables.
➢ Update or restore to factory settings.

There may be other use restrictions depending on field situations. Please consult your system integrator for further information.

5.2.3 Mode switching

5.2.3.1 About mode switching

Current mode

You can learn about the current mode of the control system by checking the mode icon on the bottom status bar of Robot Assist.

 indicates that the controller is in Manual mode and indicates that the controller is in Automatic mode. Users can click the mode icon in the HMI interface to switch between different operating modes.

Safety

For safety reasons, the system will cut off the power supply during mode switching (this means that if the system is executing an RL program and the robot is in motion, the system will trigger STOP 1).

5.2.3.2 Switching from Manual to Automatic

When to switch from Manual to Automatic

When operators need to verify the programs at all states and speeds, or when the programs are ready for full production, the system can be switched to Automatic mode.

> ⚠ **DANGER**
>
> When in Automatic mode, the robot may be triggered to move by an external signal without any warning. Before switching to Automatic mode, please make sure that the collision detection is enabled to prevent personal injury from accidental collisions between the robot and personnel!

Notes

> ⚠ **Warning**
>
> In Automatic mode, the robot can be remotely powered on and the RL program started, which means that the robot may activate at any time.
> Please consult your system integrator for the detailed configuration of the robot system.

5.2.3.3 Switching from Automatic to Manual

Switching from Automatic to Manual

Click the [icon] icon on the Robot Assist interface to switch from Automatic to Manual and see if the icon changes to [icon]. If yes, the mode is switched. If the switching fails, please troubleshoot according to the real-time log information on the top status bar.

5.3 Robot power on/off

5.3.1 Robot power-on

Power-on in Manual mode

In Manual mode, the user can power on the motor by pressing the yellow three-position enabling switch on the handheld Enabling Device and holding it in the middle position. If the sound of the robot power-on is heard or a red power-on button on the bottom status bar of Robot Assist is observed, the power-on is complete.
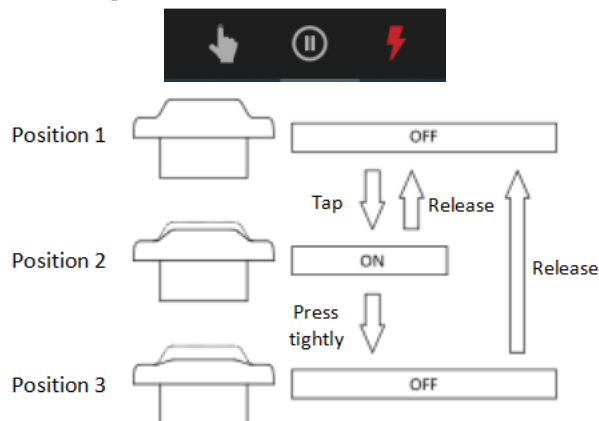
> ℹ **Notes**
>
> If the power-on fails, observe the real-time log to determine the robot's status at this time and switch the robot to a state that supports power-on before trying again.

Power-on in Automatic mode

In Automatic mode, click the Power-On button on the bottom status bar of Robot Assist to power on the motor. The method to determine whether the motor is properly powered on in this mode is the same as that in Manual mode.
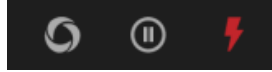
5.3.2 Robot power-off

Power-off in Manual mode

In Manual mode, the user can power off the motor by releasing or pressing the yellow three-position enabling switch all the way down to keep it in Position 1 or Position 3.

Power-off in Automatic mode

In Automatic mode, click the Power-Off button on the bottom status bar of the Robot Assist interface to power off the motor.

⚠️ Warning

In case of emergency, press the Emergency Stop button on the manual Enabling Device for emergency robot power-off. In need of power-on again, please reset the emergency stop switch manually.

---

6 Motion control

6.1 Jog mode

Jog Settings

Joint space Jog or Cartesian space Jog is available for Jog motion.

Jog motion frames available for the Cartesian space Jog include world frame, base frame, flange frame, tool frame, and work object frame.

Set Jog mode to Continuous Jog or Stepping Jog:

➢ In Continuous Jog mode, press the enabling switch to power on the robot. Then press the Jog button. The robot will move continuously at the set Jog velocity until either the enabling switch or the Jog button is released;

➢ In Stepping Jog mode, press and hold the Jog button. The robot is powered on and will move at a fixed step length; The step length can be set to precisely adjust the robot pose;

➢ Jog speed can be set to control the robot motion speed during Jog. The speed range is from 1% to 100% (100% corresponds to the robot's top TCP speed of 250 mm/s). (Both Cartesian space Jog and joint space Jog adopt TCP linear speed of 250 mm/s as the top Jog speed)

ℹ️ Notes

In Stepping Jog mode, press and hold the Jog button. Wait until the robot moves at the specified step length before releasing the Jog button. A short press may cause the robot to stop moving in advance.

Quick turn

The HMI motion interface offers convenient adjustment to common robot poses, including mechanical zero position, drag pose, shipping pose, and home pose.

The quick pose adjustment is available in Manual Mode in a way similar to Jog operation. In Manual Mode, the robot is powered on via the enabling switch. When the button for the corresponding target pose is pressed, the robot will move to the target pose in the joint space.

The motion speed can be adjusted via the Jog speed.

The Quick Turn features parameter configuration for users to use custom shipping poses, drag poses, or Home poses. Set the parameters in the Robot Configuration -> Settings -> Quick Turn page. If the custom configuration is not enabled, the default configuration takes effect.

6.2 Drag mode

Explanation

The Drag Mode of the xMate cobot is designed for users to achieve quick trajectory recording and reproduction. During programming, the robot can be positioned easily by dragging it, which substantially saves programming time.

Drag settings

The Drag Mode can be set to joint space drag or Cartesian space drag.

In the joint space Drag Mode, each axis moves independently and can be directly adjusted to reach the desired pose.

In the Cartesian space Drag Mode, two options are available: Rotate Only and Translate Only. In Rotate Only, the robot can be guided manually to rotate around TCP; in Translate Only, the robot can be guided for translational motion in the Cartesian space in different directions.

When the robot is in Manual mode and powered off, turn on the drag enabling switch on the operation panel, the robot is powered on automatically and enables Drag Mode. Press the enabling button on the end-effector drag handle simultaneously to drag the robot for point position teaching and trajectory recording.

Notes

1. Set the drag mode and drag space before enabling Drag Mode.
2. After Drag Mode is enabled, the robot will be powered on automatically. In this case, the drag mode and drag space cannot be set. Please disable Drag Mode before setting them.

Warning

1、 Before enabling Drag Mode, please ensure the robot's dynamic parameters and load parameters are set accurately. Otherwise, a failure may occur when enabling Drag Mode, or the robot may float during dragging.

2、 Set the parameters using the dynamic parameter identification function and the load identification function provided by the system.

DANGER

Before using Drag Teaching, please ensure the following parameters are set correctly:
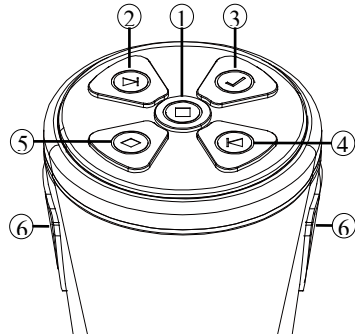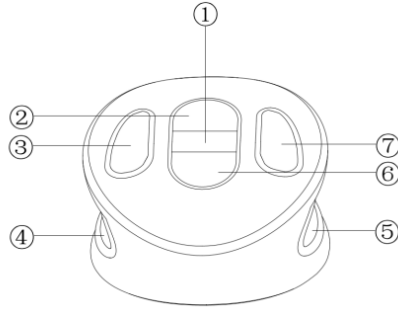
1.	Robot model;
2.	Robot installation method: floor mounting or ceiling mounting;
3.	Dynamic parameters of the robot and its load;
4.	Mechanical zero calibration;

Otherwise, when the angles of axes are in the wrong state, the controller may not be able to calculate the correct output torque, and the robot cannot work in Drag Mode properly.

Applicable models:

The Drag Mode and its extended functions (end-effector handle, point position teaching, continuous trajectory teaching, and trajectory reproduction) are only available for xMate cobots.

6.2.1 End-effector handle

Explanation

Series	Introduction	Definition
xMate ER	The xMate ER series robot end-effector integrates a Pilot handle with an intelligent interactive panel. In Drag Mode, the buttons on the Pilot handle can be used for quick point position teaching and continuous trajectory teaching, providing better human-machine interaction.	Definition of buttons on end-effector Pilot handle:
xMate CR	The xMate CR series robot end-effector integrates an xPanel handle with an intelligent interactive panel. In Drag Mode, the buttons on the Pilot handle can be used for quick point position teaching and continuous trajectory teaching, providing better human-machine interaction.	Definition of buttons on end-effector xPanel handle:

Definition (xMate ER):

No.	Definition
1	Update the teaching point with the current pose, start/stop trajectory recording
2	Next
3	Add the point position/trajectory in the list, confirm pop-up window prompts
4	Previous
5	Delete the point position/trajectory in the list, cancel pop-up window prompts
6	In Drag Mode, press the two enabling buttons at the same time to activate the drag function

Definition (xMate CR):

No.	Definition
1	Update the teaching point with the current pose, start/stop trajectory recording
2	Moves forward
3	Delete the point/track in the list and cancel the pop-up prompt
4	In Drag Mode, press the two enabling buttons at the same time to

5	activate the drag function	
6	Moves backward	
7	Add the midpoint/track to the list and confirm the pop-up prompt	

6.2.2 Point position teaching

Explanation

Turn on the drag enabling switch on the operation panel, and the robot is powered on automatically and enables Drag Mode. The following operations can be performed through Robot Assist and the robot end-effector drag handle:

	Operation	Description
1	Create/load a project and enter the Point List interface	The end-effector buttons only respond when the current page of Robot Assist is Point List or Path List.
2	Press the two enabling buttons on the end-effector handle at the same time, drag the robot to any position, and release the drag enabling button. Press the Add Point button on the end-effector handle.	A new teaching point of the current pose is added to the end of the Point List, and the cursor is now at the new teaching point.
3	Press the Previous/Next button on the end-effector handle	Move the cursor to the previous/next point in the Point List and select the point
4	Select a point to update in the Point List, drag the robot to another position, and release the drag enabling button. Press the Update Point button on the end-effector handle.	The selected point in the Point List is updated with the current pose.
5	Select a point to delete in the Point List. Press the Delete Point button on the end-effector handle and confirm.	A pop-up window prompt will appear when you try to delete a path. If you press the OK button on the end-effector handle, the selected path will be deleted from the Path List. If you press the Cancel button on the end-effector handle, the pop-up window will be closed, and the selected path will remain on the Path List.

6.2.3 Continuous trajectory teaching

HMI operation

Turn on the drag enabling switch on the operation panel, and the robot is powered on automatically and enables Drag Mode. The following operations can be performed through Robot Assist:

Step 1: Create/load a project, move the robot to any start position, and enter the Project -> Path

interface.

Step 2: Click on the Path List interface to create a new path. Enter the sub-interface for new path settings.
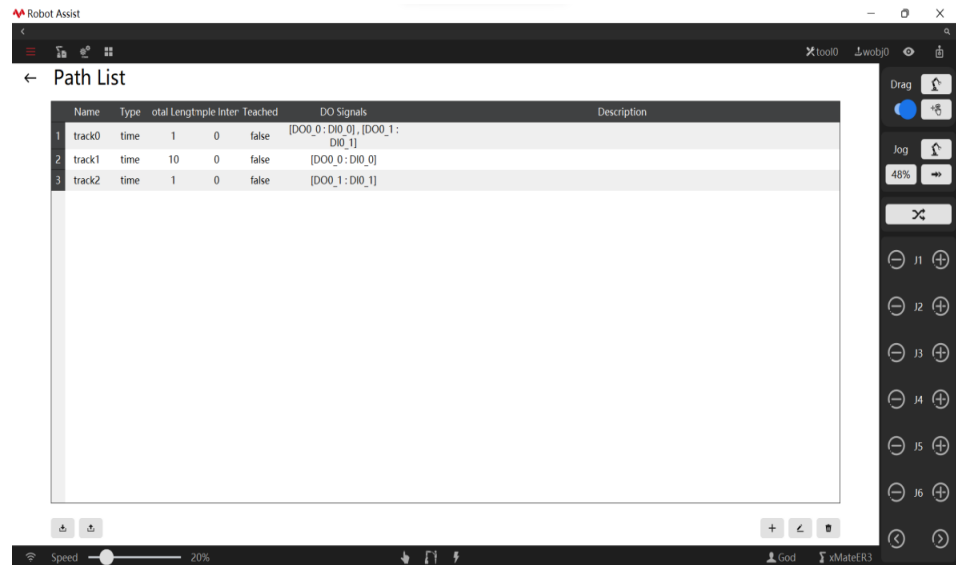
Step 3: Set the new path name, description, and total recording time. To record DO signals, click the Select DO button and select DO signal in the pop-up window. A DI mapping signal can be set for each DO signal, where the change of the DI signal will be recorded as the change of the DO signal and be output to the DO signal when the path is recorded. If the DI signal is not associated with the DO signal, the change of the DO signal will be recorded directly, and the output of the DO signal can be manually set on the Status Monitoring - IO Signal interface.
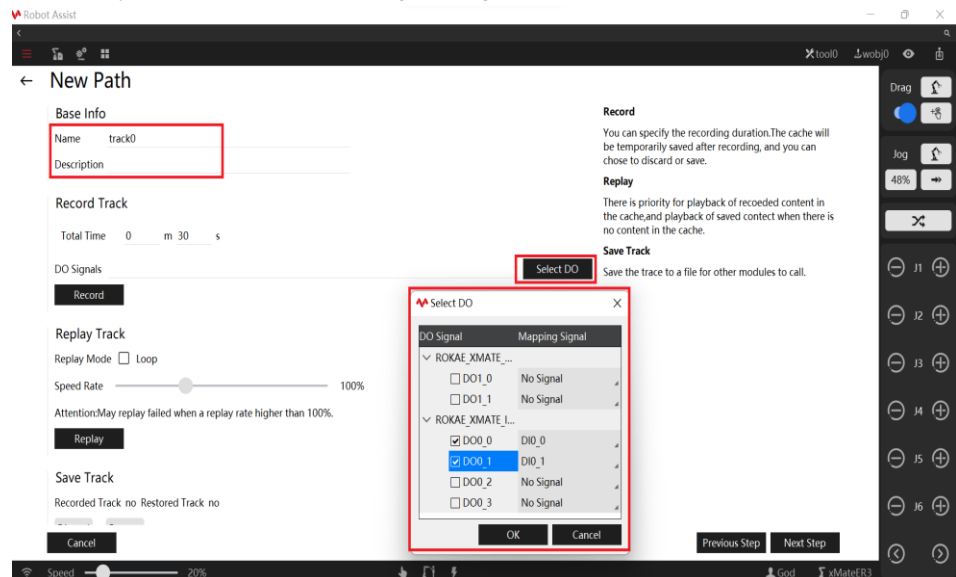
Step 4: After the parameters are set successfully, click the Start Recording button to start trajectory recording. During recording, drag the robot within the countdown time and set relevant DIO signals to complete trajectory recording. If the Stop button is pressed during recording, the trajectory before the stop time is recorded.
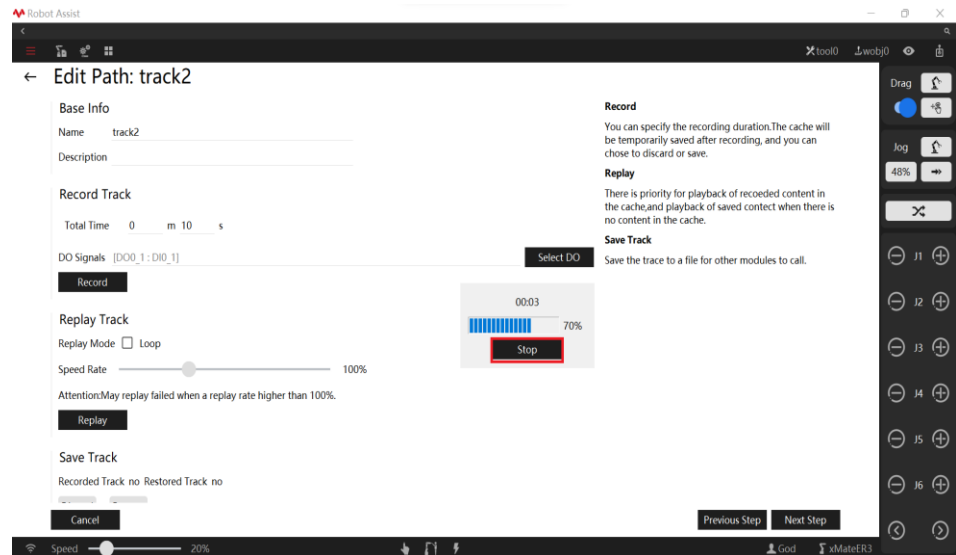
Notes

The recorded trajectory is temporarily saved in the cache. You can discard it or save it as a file. Recorded Trajectory Available/Unavailable is displayed on the page.

End-effector button operations

Turn on the drag enabling switch on the operation panel, and the robot is powered on automatically and enables Drag Mode. The following operations can be performed through Robot Assist and the robot end-effector drag handle:
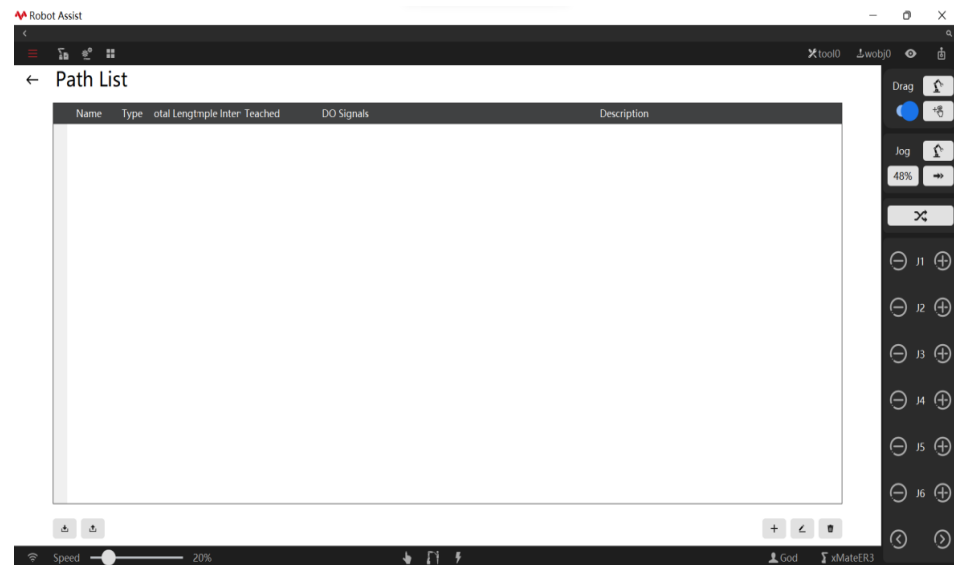
	Operation	Description
1	Create/load a project, move the robot to any start position, and enter the Path List interface.	The end-effector buttons only respond when the current page of Robot Assist is Point List or Path List.
2	Press the Add Path button on the end-effector handle	A new path is added to the end of the Path List, and the cursor is now at the new path.
3	Press the Previous/Next button on the end-effector handle	Move the cursor to the previous/next path in the Path List and select the path

4	Select a path in the Path List to start recording. Press the Start Trajectory Recording button on the end-effector handle and press the two enabling buttons on the end-effector handle at the same time to drag the robot for trajectory recording.	The trajectory recording starts after the Start Trajectory Recording button is pressed. Press the Stop Trajectory Recording button to stop recording, and the trajectory is saved automatically.
5	Select a path to delete in the Path List. Press the Delete Path button on the end-effector handle and confirm.	A pop-up window prompt will appear when you try to delete a path. If you press the OK button on the end-effector handle, the selected path will be deleted from the Path List. If you press the Cancel button on the end-effector handle, the pop-up window will be closed, and the selected path will remain on the Path List.

6.2.4 Trajectory reproduction

Explanation

After a successful continuous trajectory teaching, the trajectory is recorded. Playback the recorded trajectory on the recording interface and confirm, and then save it manually after confirmation.

Playback settings

Check Loop in the playback mode for looped playback.

The playback speed can be set between 1% and 300%.

> **i** Notes
>
> It is recommended to set the playback speed between 1% and 100%. When the playback speed is greater than 100%, a drive following error might occur.

Playback operation

Step 1: Turn off the drag enabling switch and switch to Automatic mode. Click the Power-on button to power on the robot.

Step 2: Click Playback for the robot to play the recorded trajectory.

Step 3: Confirm the trajectory after the trajectory playback is finished. Click Save for the recorded trajectory to be saved as a file successfully.

Step 4: After completing the trajectory recording, trajectory confirmation, and trajectory save, click Next to return to the directory list that displays the directory of the saved trajectories.

7 Robot Configuration

7.1 Basic settings

7.1.1 User groups and permissions

User levels

The xCore system is built-in with three user levels, namely Operator, Admin, and God, with the operation permissions ranking from low to high.

Switching from a low-privileged user to a high-privileged user requires a password, which is 123456 by default. Otherwise, it is not required. A user of a higher permission level can modify the password of a same- or lower-level user. The password of an Operator-level user cannot be modified.

Division of operational authority

Category	Function	Operator	Admin	God
Project	Project management	N	Y	Y

	(Create, Import, Export)			
	View project (including program and object data such as IO and variables)	Y	Y	Y
	Edit project (including program editing and object settings such as tools)	N	Y	Y
Robot motion and program running	Mode switching	N	Y	Y
	Power on/off	N	Y	Y
	Start/stop program	Y	Y	Y
	Adjust program running speed	N	Y	Y
	Single-step program debugging	N	Y	Y
	APP running	Y	Y	Y
	Jog/drag interface	N	Y	Y
Running interface	Set auto-load project	N	Y	Y
	View runtime data	Y	Y	Y
System settings	System upgrade	N	Y	Y
	Data backup/recovery	N	Y	Y
	User permission management	N	Y	Y
	Function authorization	N	Y	Y
	System time setting	N	Y	Y
	System language setting	N	Y	Y
	Controller reboot	N	Y	Y
Robot settings	Body parameter setting	N	N	Y
	Robot installation	N	Y	Y
	Zero Calibration	N	Y	Y
	Motion parameter recognition	N	N	Y
	Extended IO module configuration	N	Y	Y
	System IO setting	N	Y	Y
	End tool setting	N	Y	Y
	Socket setting	N	Y	Y
	Safety setting	N	Y	Y
	Clear servo alarms	N	Y	Y
	RCI function setting	N	Y	Y
Log management	View log	Y	Y	Y
	Delete log	N	Y	Y
	View/delete Debug log	N	N	Y
	Log backup	N	Y	Y
Help interface	View Help	Y	Y	Y
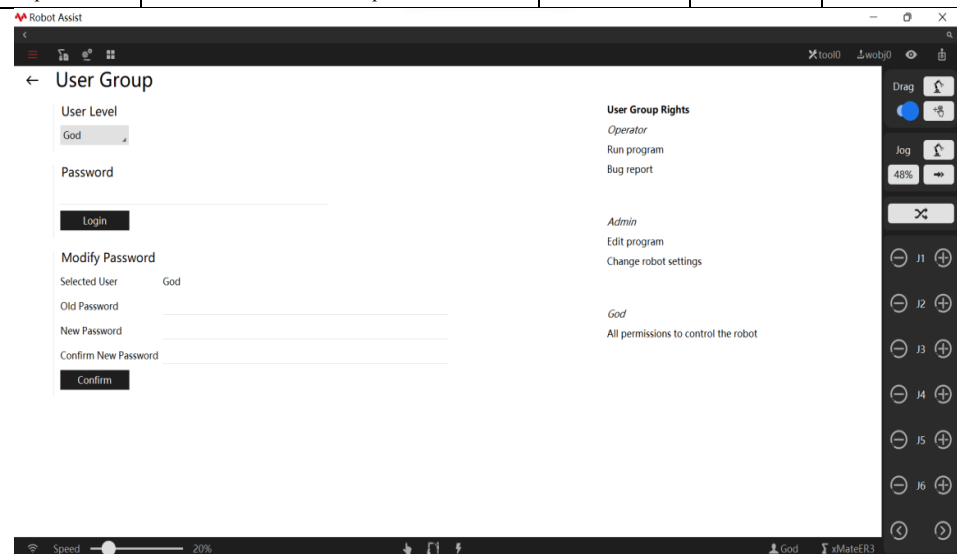
7.1.2 Controller settings

System information

Controller setting options are available in the xCore system to soft reboot or shut down the controller. You need to save all configuration information before restarting. If the controller is shut down, the controller software can only be restarted after the control cabinet is powered off and then powered on again.

System configuration

The robot type, controller cabinet type, and safety board type should be properly configured to ensure the robot works normally.
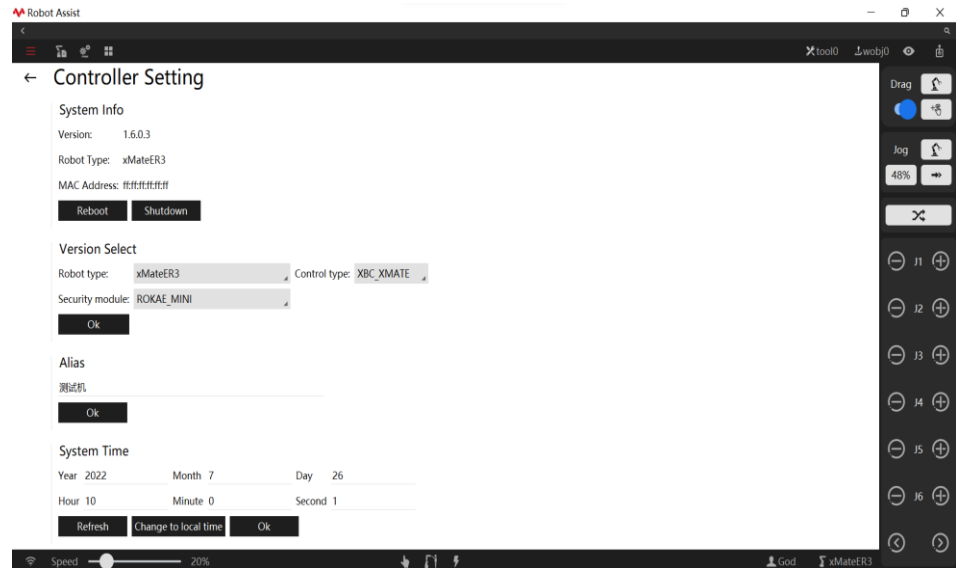
Alias

Set an alias for each controller, so that the robots in the same LAN can identify the controllers conveniently. The alias will be displayed on the interface when the robot searches for controllers, as shown in the figure below.
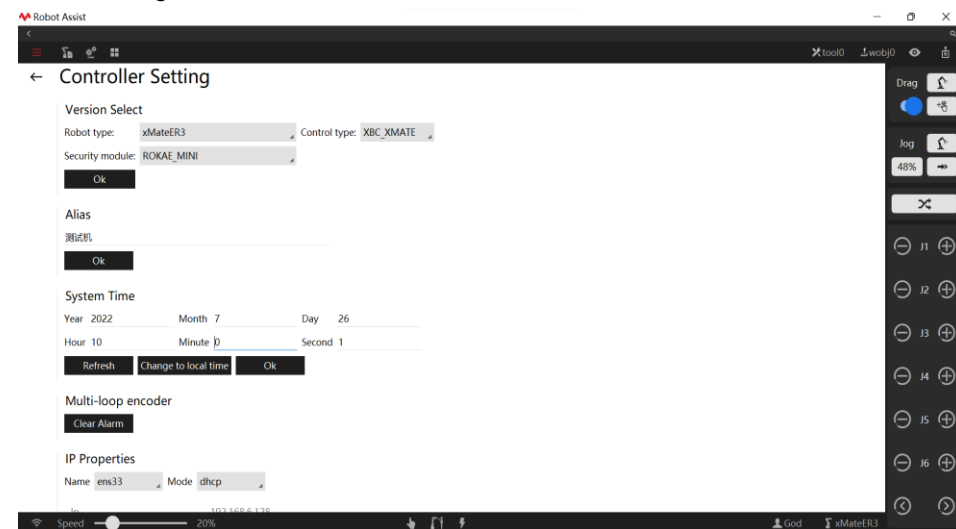
System time

The system time shows the system time of the controller.
The system time provides an absolute time reference for functions such as logging to trace the moment of relevant events.

Click [Refresh] to check the robot's time reference to see if it is consistent with the system time. The controller system time can be modified manually, or it can adopt the current system time of the device on which Robot Assist is running. Users can directly modify the controller system time manually or click [Change to local time] to adopt the current system time of the device on which Robot Assist is running as the controller system time.

When the system time displayed on Robot Assist is not consistent with the system time in the lower right corner of Robot Assist, the user can click [Change to local time] to update the controller system time with the system time of the device on which Robot Assist is running.
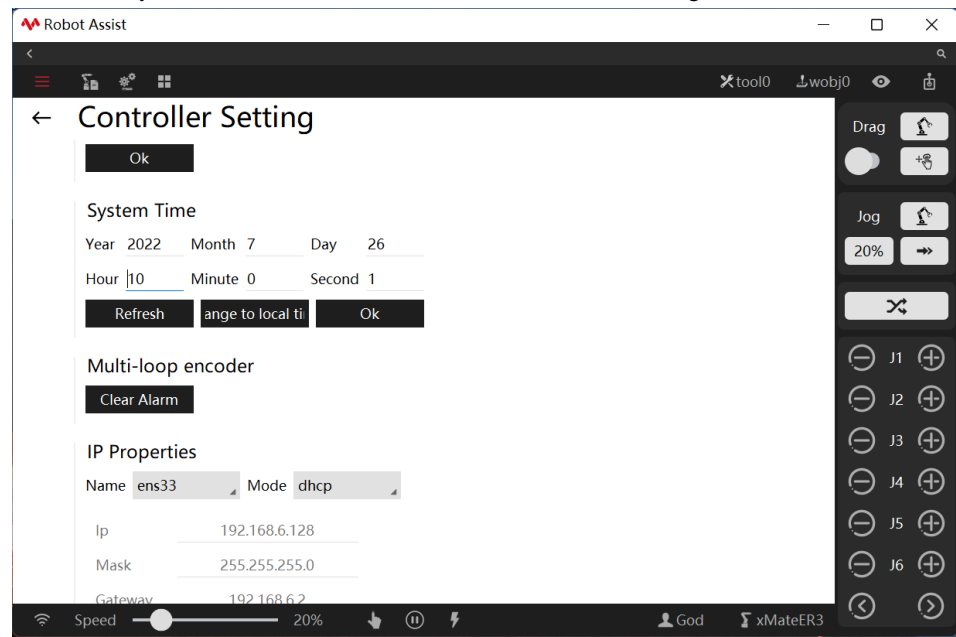
⚠	Warning
	1. The system time is the absolute time standard for log information. Do not modify it arbitrarily. Wrong system time will make it impossible for the user to trace the moment of a relevant event through the log.
	2. Do not frequently perform the two operations - Obtain controller time or Set to current time. The interval between two operations (either one or both) should be greater than 5 seconds.

Multi-loop encoder

Clear multi-loop error messages on the encoder.

ℹ	Notes
	1、 This function is only available for industrial robots.
	2、 After replacing the encoder battery of the robot, use this function to clear error messages before re-calibration.

System IP properties

Set the connection mode of the robot's external network interface. For details, refer to Chapter 4 Connecting to the Robot.

Log save levels

Set the log save level. There are three levels of log - "info", "warning", and "error", ranking from low to high. Set the level from which the log is kept. The log of lower levels will only be displayed

online, and will not be kept in the log.
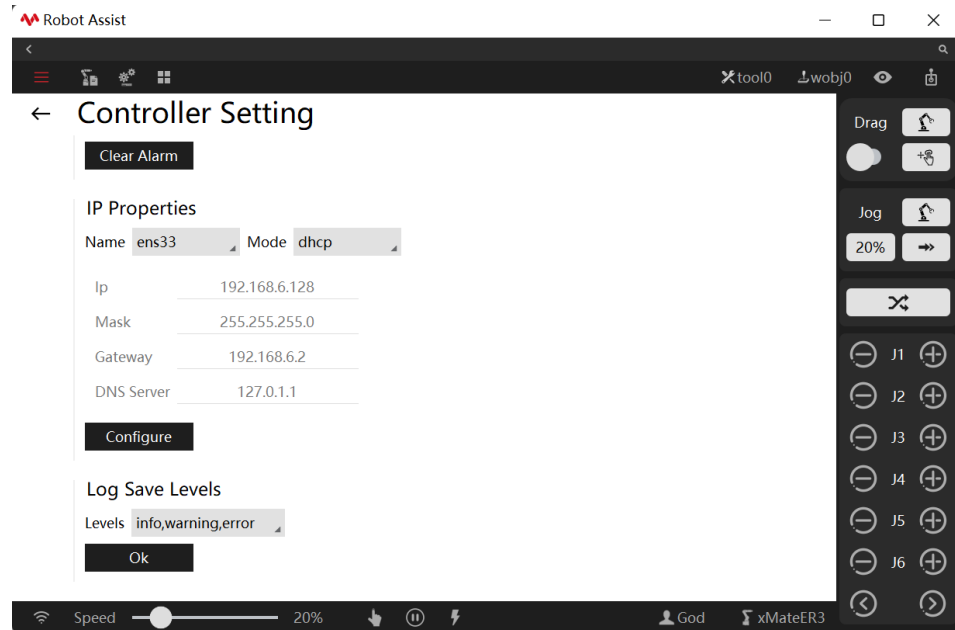
For example, if "warning,error" are selected, the "info" level log will only be displayed when it appears, and cannot be queried in Diagnosis or after the controller is restarted. The log of the "warning" and "error" level will be displayed online, and can also be queried in Diagnosis or after the controller is restarted. Query the history in Diagnosis -> Controller Log.

7.1.3 Zero Calibration

Explanation

The xCore system provides robot calibration, including mechanical zero calibration and force sensor zero calibration. The calibration can be performed by "One-Key Calibrate" or Single Joint Calibration;
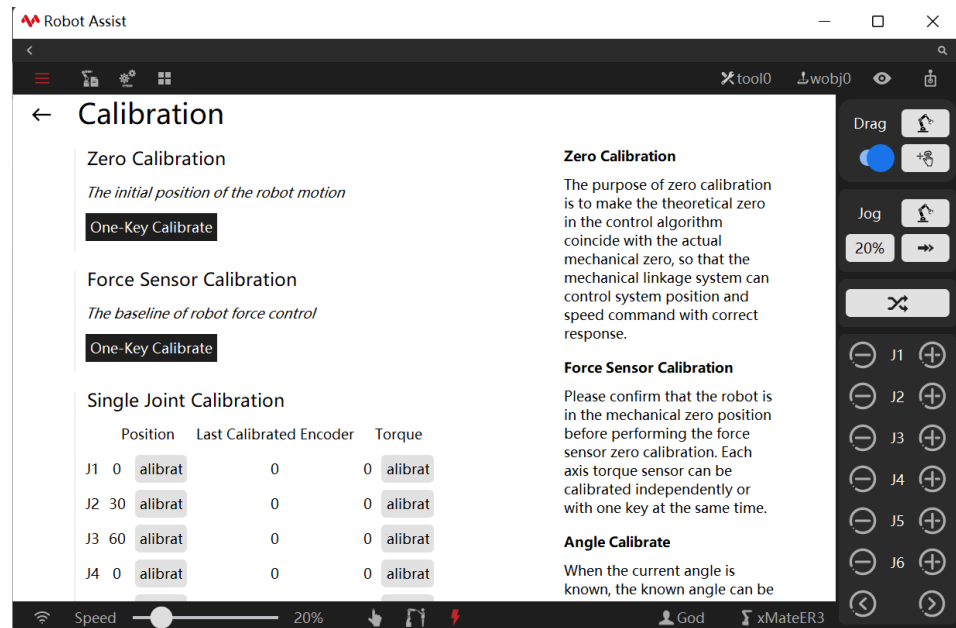
Mechanical zero calibration

The purpose of mechanical zero calibration is to coincide the theoretical zero point in the control algorithm with the actual mechanical zero so that the mechanical linkage system can make correct responses to position and speed commands of the control system.

More generally, the zero calibration is to use certain pre-designed positioning devices on the mechanical body to rotate the joints of the robot to a specific angle, and notify the control system of recording the value of each joint motor encoder at this time.

> ⚠️ Warning
>
> 1、 The mechanical zero point is the theoretical zero point in the robot control algorithm. Please do not calibrate it arbitrarily and ensure that all robot joints are at the zero point using the mechanical zero calibration block before calibration.
>
> 2、 Do not perform the mechanical zero calibration on the robot after it is calibrated by a laser tracker. Otherwise, the zero point calibrated by the laser tracker will be lost, therefore affecting the robot accuracy. In case the zero point of the robot is lost, please contact ROKAE to restore the zero point.

Torque zero calibration

The purpose of the force sensor zero calibration is to coincide the theoretical joint torque zero with the actual joint torque zero so that the mechanical linkage system can correctly capture the actual torque of the joints. Put simply, the force sensor zero calibration is to move the robot's joints to a specific location unaffected by gravity and notify the control system of recording the value of each joint force sensor at this time.

> ⚠️ Warning
>
> Torque zero calibration can also be performed in a non-mechanical zero; for optimal calibration accuracy, it is recommended to set all joints to the mechanical zero before torque zero calibration.

Dynamic calibration of the torque sensor

During robot motion, zero drift is inevitable for torque sensors, which may cause the robot to float during dragging. In the case of zero drift, dynamic calibration can be enabled. When the force control-related commands such as enable drag, or force control are turned on in the RL program, the system will automatically zero calibrate to ensure that force control-related functions can be used normally.

⚠️ Warning

Dynamic calibration involves two risks:

1. If the robot is in contact with the environment during dragging, i.e., the robot is in a non-free state, the calibrated zero may have a big error, which may result in the wrong torque calculated and failure to enable force control;

2. The robot may drift during dragging at certain positions after a torque sensor zero calibration is performed when at a non-mechanical zero position.

For these reasons, dynamic calibration should not be turned on unless the torque sensor zero sees serious drifting.

This function is turned off by default.

Angle calibration settings

Due to space constraints in certain scenarios, the robot cannot return to the mechanical zero. In this case, angle calibration is used to calibrate the robot. Angle calibration is to input the known current angle to calibrate the robot, achieving the same result as the mechanical zero calibration. Take the xMate7 Pro seven-axis robot as an example based on the assumption that there are obstacles in the 4-axis space and the robot cannot return to the vertical state of the mechanical zero, jog the 4-axis to 90 degrees individually to perform zero calibration. Input the current angle in Angle Calibration to perform the "mechanical zero calibration".
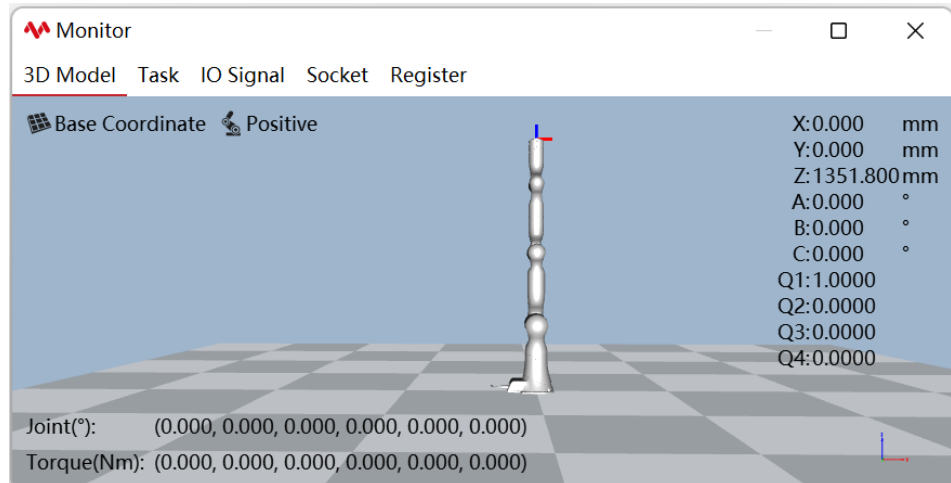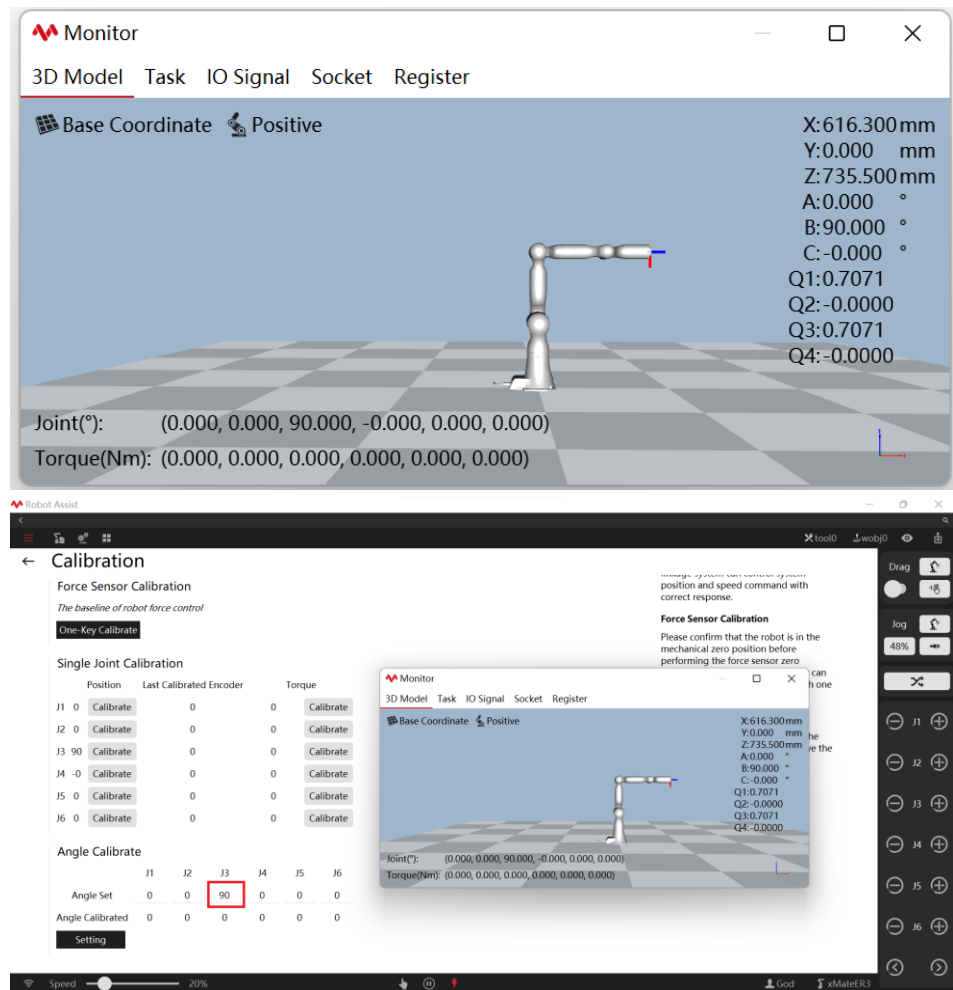
Please note that in the above example, although it is calibrated in a different orientation, the zero of the robot remains in a vertical state. Therefore, if you directly use the 🔔 Quick Turn to Zero function after a successful angle calibration by inputting the current angle of the 4-axis at 90 degrees, the robot will still move to the vertical state of the mechanical zero and thus collide with the obstacles! So bear in mind that the Angle Calibration function calibrates the zero. It does not mean that the zero is at the current angle.

7.1.4 Base calibration

What is the base frame?

The base frame at the center of the robot base is described relative to the world frame to confirm the placement position of the robot. The base frame should be calibrated when the robot is installed at any angle or there are multiple robots.

Base calibration
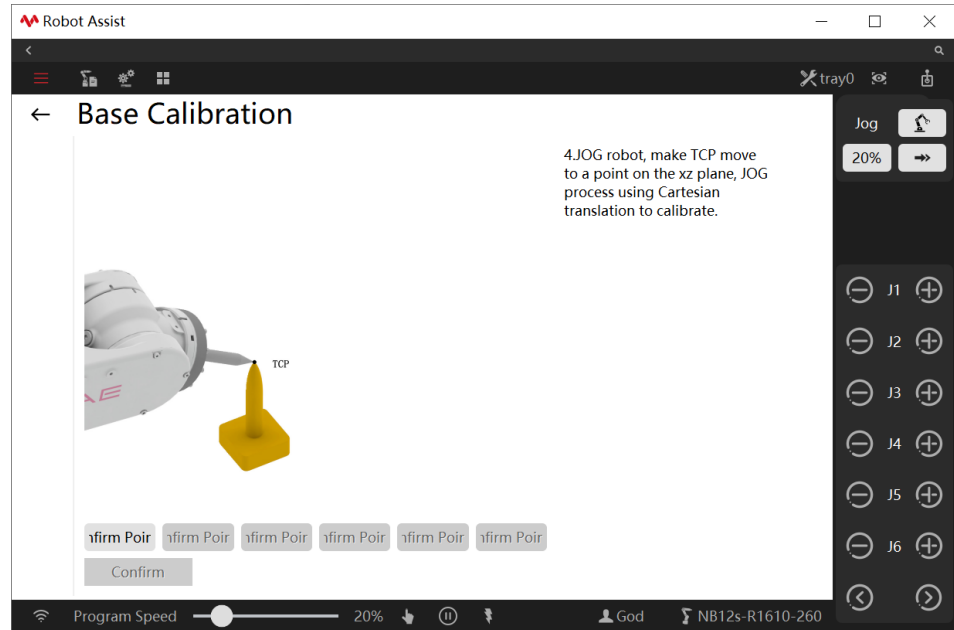
As shown in the picture above, the general steps to calibrate the base frame are as follows:

	Operation	Description
1	Use admin to log in to the system and calibrate the tool frame.	The selected tool should be consistent with the tool installed on the flange.
2	Confirm the calibration method.	The system supports the six-point method (default) and manual input. If there is a known offset of the base frame relative to the world frame, it is advised to use manual input.
3	Define the position of the auxiliary point.	When the tool is too far to reach the world frame. The base frame can be calibrated by auxiliary position. The auxiliary position is defined according to the world frame.
4	Jog to confirm each teaching point in turn.	Users could choose whether to save the base frame data according to the calibration result.

Manual input
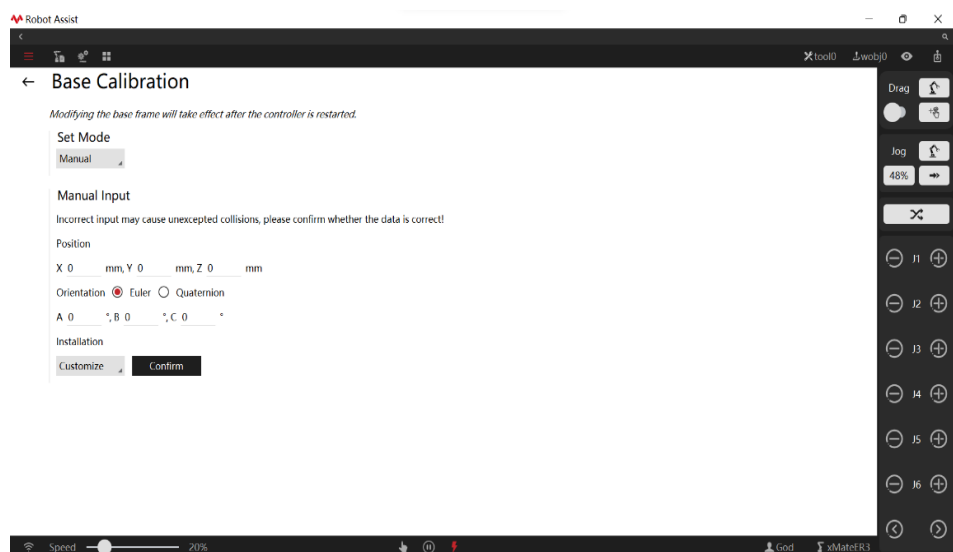
The base frame can be set by manual input. Manually input the position and orientation of the base frame relative to the world frame. The orientation can be specified with Euler angles or quaternions.

Mounting method	Description	A	B	C
Floor mounting		0	0	0
Wall mounting A	Power cable outlet is above the base	0	90	0
Wall mounting B	Power cable outlet is on the right of the base	-90	0	-90
Wall mounting C	Power cable outlet is under the base	180	-90	0
Wall mounting D	Power cable outlet is above the base	90	0	90
Ceiling mounting				

⚠ Warning

When manually inputting parameters to calibrate the base frame, make sure that the data is accurate. Incorrect parameters may lead to unintended collisions.

7.1.5 Dynamic settings

Explanation

The dynamic settings page is used to set the dynamic model parameters of the robot. The dynamic model is mainly used for functions such as robot force control, drag teaching, virtual wall, and collision detection. Please ensure the robot's dynamic model parameters are correctly set. Otherwise, the above functions may not work properly or may cause the robot to shake abnormally.
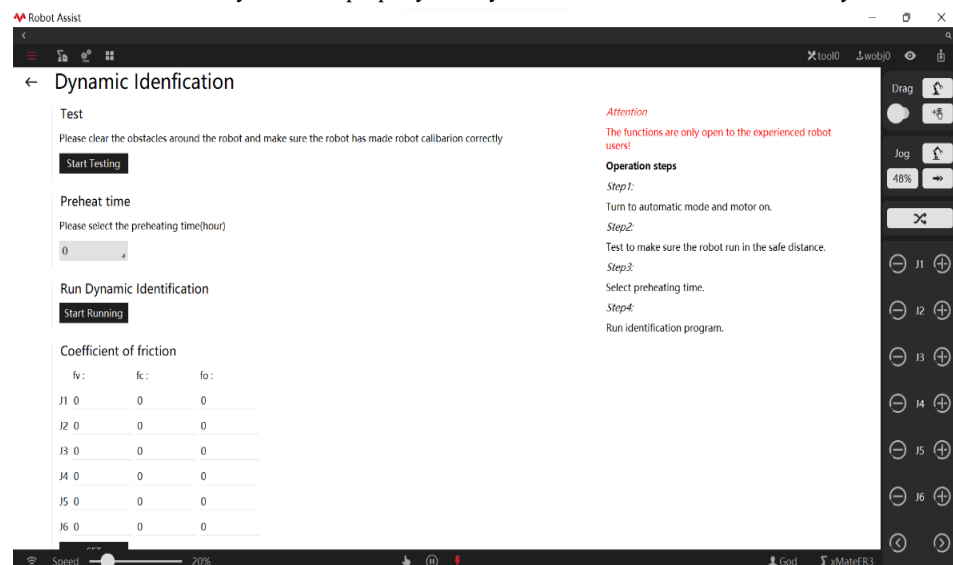
Dynamic parameter identification

Dynamic identification allows the robot to execute a series of preset trajectories and collect info during the motion to calculate the body dynamic parameters required.

Step 1: Robot Configuration -> Settings -> Dynamic Settings

Step 2: Remove the obstacles around the robot. Make sure there are no obstacles (except the base) in the **reachable area** of the robot. For details about the reachable area of the robot, refer to the robot installation manual of each model.

Step 3: Preheat time is the continuous running time of identification. The longer the running time, the better the identification performance. You can set the preheat time to 0, 1, 2, or 4 hour(s). If the preheat time is set to 0 hour, the identification is finished after the robot completes a full trajectory, which lasts about 1 minute.

Step 4: Click **Start Running**. The robot will now automatically execute the dynamic parameter identification program.

Step 5: Wait for the identification result. If it shows the identification is successful, that means the identification is finished normally. If it shows the identification failed, refer to the **Error handling** section below.

Use restrictions

1. The dynamic parameter identification function is not available for XB12s-3 and XB12s-4.

2. Please ensure no obstacles exist in the reachable area of the robot.

3. Please ensure the robot zero calibration is correctly performed before use. For details about the robot zero calibration, refer to Chapter 7.1.3.

4. Dynamic identification is not allowed when the robot is loaded.

5. The identification result will only take effect after the robot restarts.

Error handling:

1. Click the Stop button on Robot Assist to stop the identification process.

2. In case of emergency, press the emergency stop button to stop the robot immediately.

3. The identification result will not be recorded if it is interrupted. Re-execute the program for identification.

Friction identification

Like dynamic parameter identification, friction identification allows the robot to execute a series of preset trajectories and collect info during the motion to calculate the friction parameters required. Dynamic identification and friction identification are two independent features, and their orders are free.

Step 1: Robot Configuration -> Settings -> Dynamic Settings

Step 2: Turn off dynamic constraint and dynamic feedforward. For dynamic constraint and dynamic feedforward switches, refer to the section below;

Step 3: The robot will now automatically execute the friction identification program after clicking **Start Running**.

Step 4: Wait for the identification result. If it shows the identification is successful, that means the identification is finished normally. The friction coefficient will only take effect after a restart, and the identified friction coefficients for each axis are displayed on the current interface. If it shows the identification failed, refer to the **Error handling** section below.

Use restrictions

1. Please ensure no obstacles exist in the reachable area of the robot.

2. Please ensure the robot zero calibration is correctly performed before use. For details about the robot zero calibration, refer to Chapter 7.1.3.

3. Friction identification is not allowed when the robot is loaded.

4. The friction identification result will only take effect when the robot restarts.

Error handling:

1. Click the Stop button on Robot Assist to stop the identification process.

2. In case of emergency, press the emergency stop button to stop the robot immediately.

3. The identification result will not be recorded if it is interrupted. Re-execute the program for identification.

4. In case of abnormal friction identification result, the nominal value is used, and a prompt is displayed on the interface. If the nominal value is inappropriate, the friction coefficient can be modified manually on the interface.

Friction settings
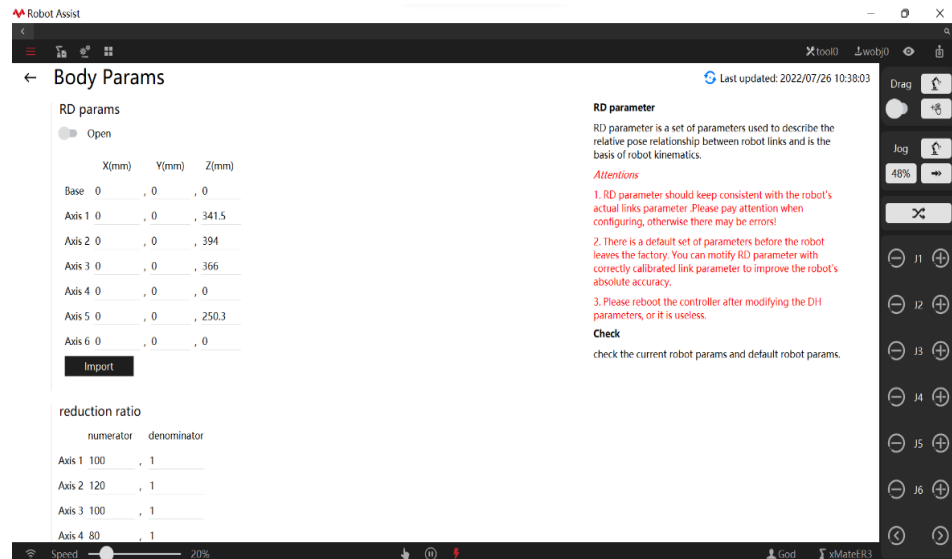
The friction coefficient page displays the friction coefficients of the robot, including viscous friction coefficient fv, Coulomb friction coefficient fc, and Coulomb friction coefficient bias fo. If friction identification is not performed, nominal friction coefficients are displayed. If performed, the factory identity friction coefficients are displayed. Friction coefficients can be modified manually and take effect after restart. But users are not recommended to modify these parameters as the dynamic functions may go wrong.

The third-order friction coefficient is an advanced function of the controller. The internal parameter could not be modified by users.

Dynamic feedforward switch

1. The dynamic feedforward switch determines whether the controller turns on or off the dynamic feedforward function and is turned on by default.

2. Users are not recommended to turn off the dynamic feed-forward function by themselves, which may cause jitter when power on and worse trajectory accuracy.

3. The dynamics feedforward should be turned off in certain situations, including base frame calibration when the robot adopts wall/ceiling mounting and friction identification.
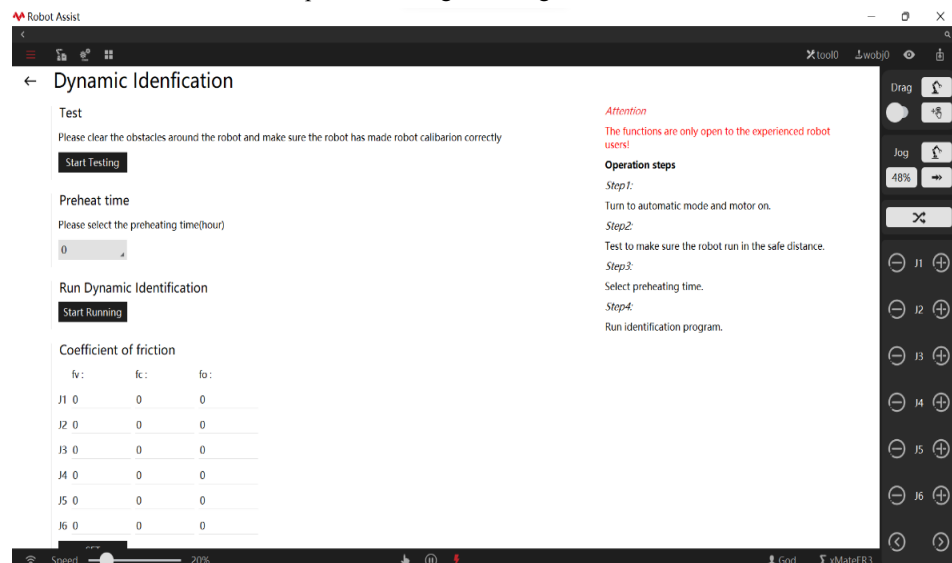
Dynamic constraint switch

1. The dynamic constraint switch determines whether the controller turns on or off the dynamic

constraint function and is turned on by default.

2. Users are not recommended to turn off the dynamic constraint function by themselves, which may cause motor overload or abnormal shaking.

3. When the dynamic constraint switch is turned on, two options including "Nominal Dynamic Params" and "Factory Identify Dynamic Params" are available. "Nominal Dynamic Params" means nominal parameters will be used in the dynamic control. The same models using the "Nominal Dynamic Params" will deliver the exact same motion velocity and takt time when executing the same motion program, yet the motion performance may be weaker, or there may be motor overload. When "Factory Identify Dynamic Params" is selected, the robot will be in the best dynamic control status for the shortest takt time allowed, and the motor will be protected from overload. But robots running the same motion program may be slightly different in velocity and takt time.

7.1.6 Body parameters

Explanation

Body parameters include RD parameters, reduction ratio, and coupling coefficient. These parameters are all related to the robot body, including the properties of its mechanical mechanism and components. The parameters on this page directly affects the accuracy of the robot motion. Please modify them with discretion.

RD parameters

RD parameters are a set of parameters used to describe the relative pose relationship between the robot's link frames. They are the foundation for robot kinematics.
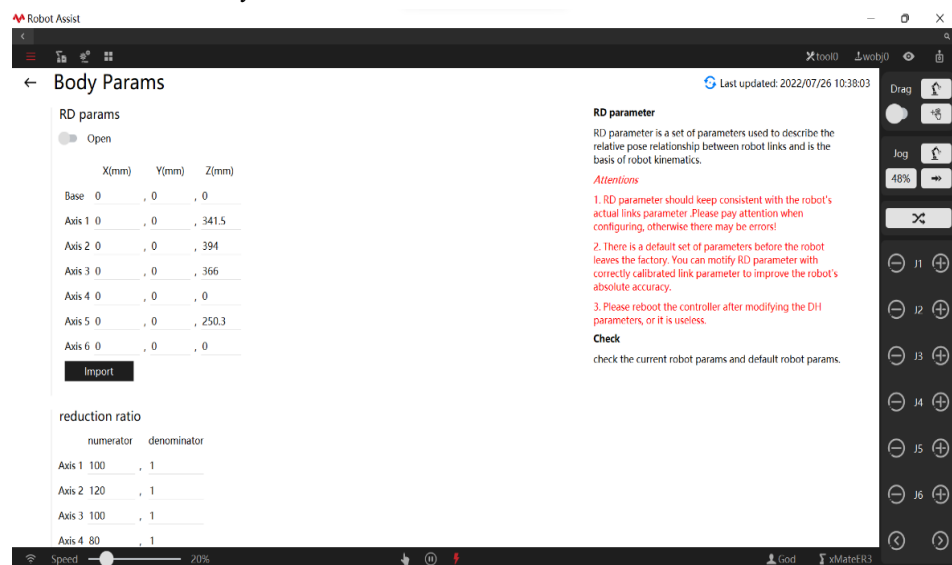
- ➤ The RD parameters need to match the actual link parameters of the robot. Please configure the parameters with caution. Otherwise, errors such as "exceeding the workspace" may occur.
- ➤ A set of parameters is configured as default settings before delivery. After calibrating the robot's link parameters properly, modify the RD parameters to increase the robot's absolute accuracy. Check the rationality of the calibrated RD parameters before modifying or importing parameters;
- ➤ Restart the controller for the modified RD parameters to take effect.

Reduction ratio

The reduction ratio is the parameter of the reducer in each axis of the robot. Do not modify the factory settings. Configure the reduction ratio according to the manufacturer's instructions only after replacing the reducer with a different model.

Coupling coefficient

The motion of the Axis 4, Axis 5, and Axis 6 of the robot is coupled. The coupling coefficient is used to describe the coupled motion of other joints when these joints move. Do not modify the factory settings. Modify the parameter according to the manufacturer's instructions only after replacing with it a different model of reducer or driving element.

Use restrictions

Only God users can modify the parameter.

7.1.7 Kinematic parameters

Explanation

The kinematic parameters include the maximum speed, maximum acceleration, and maximum acceleration jerk of each axis of the robot. The kinematic parameters affect the maximum speed, maximum acceleration, and maximum acceleration jerk that the robot can achieve during motion, as well as its takt time and smoothness. A set of parameters is configured as default settings before delivery. Modifying the kinematic parameters may cause the robot to shake abnormally, report errors, or reduce its service life. Please modify them with discretion.
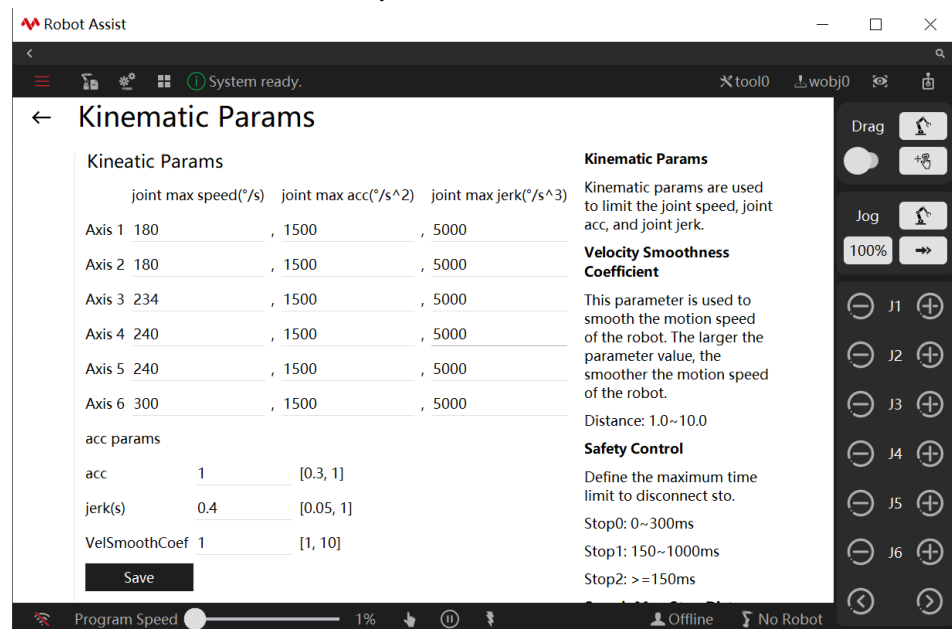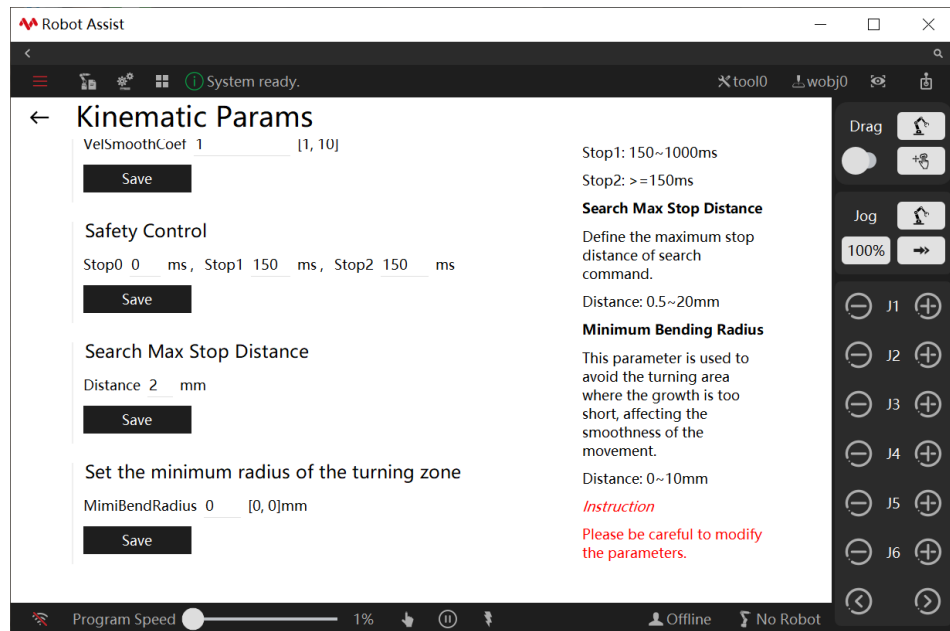
➤ Maximum Axis Velocity: the maximum velocity allowed for each axis during robot motion, mainly limited by the motor velocity.

The factory parameters are generally adopted, and they do not require modification.

➤ Maximum Axis Acceleration: the maximum acceleration allowed for each axis during robot motion, mainly limited by the motor torque. The parameter takes effect when dynamic constraint is turned off, and it limits the maximum acceleration for each axis during robot motion; when the dynamic constraint is turned on, the parameter becomes invalid, and the maximum acceleration for each axis during robot motion is calculated through the dynamic model.

The maximum acceleration set should be no less than 3-5 times the maximum velocity of the axis.

➤ Maximum Axis Jerk: the maximum jerk allowed for each axis during robot motion. Jerk is the derivative of acceleration to time. In most cases, the higher the jerk, the more likely the robot may shake during the motion, and vice versa. The actual jerk increases when the robot passes through the turning zone. In this situation, the jerk has an obvious impact on the robot takt. When the program involves many small turning zones, the jerk can be increased appropriately to speed up the takt. Additional attention needs to be paid to the robot's shaking. The maximum jerk set should be no less than 3-5 times the maximum acceleration of the axis.

➤ Acceleration Multiplier: This parameter is used to scale the robot's acceleration during operation. The larger the value, the higher the robot acceleration, and vice versa.

➤ Acceleration Rise Time: The time for the robot acceleration to increase from the minimum to the maximum. The smaller the value, the faster the robot accelerates, and vice versa.

➤ Velocity Smoothing Factor: This parameter is used to smooth the robot's velocity in the turning zone. The larger the value, the less the robot slows down in the turning zone, and vice versa. When the value is set to 1.0, the velocity is not smoothened when the robot passes through the turning zone. The larger the value, the more likely the robot is to shake in the turning zone.

This parameter is used to push the robot's ultimate performance. During commissioning, firstly check how badly the robot is shaking when the parameter is set to 1.0. If the robot shakes violently, the robot has reached its limit and there is no need to increase the value. If the robot runs smoothly but the velocity drops severely when it passes through the turning zone, this parameter can be gradually increased to make the motion smoother while observing the robot's running status. This parameter can be increased by 0.1-0.5 each time.

➤ Safety Control: The time from the receipt of the stop signal to the full stop of the robot. The

smaller the value, the faster the robot stops, and vice versa.

➢ Search Max Stop Distance: When a Search command is used, the distance traveled by the robot TCP from the receipt of the stop signal to the full stop of the robot shall not exceed this value.

➢ Set the minimum radius of the turning zone: The shortest turning zone allowed that can be specified by the turning zone radius. This parameter can be used to avoid generating a turning zone too short and to make motion smoother. When the control system detects that the length of a trajectory is below the set value of this parameter and the trajectory needs to generate a turning zone, the control system will automatically combine the trajectory and the adjacent trajectories into one trajectory and generate a turning zone with an appropriate length. The larger the value, the longer the minimum turning zone and the smoother the robot passes through the turning zone. When this parameter is set to 0, the control system strictly follows the parameters to generate the turning zone.

Use restrictions

Only God users can modify the parameter.

7.1.8 Force control parameters

Explanation

Force control parameters are force control-related parameters adapted to the actual hardware equipment and environment.

Important parameters can be adjusted and switched on the HMI interface. Two sets of control parameters built into the controller can be flexibly selected based on the usage and the actual application scenario.
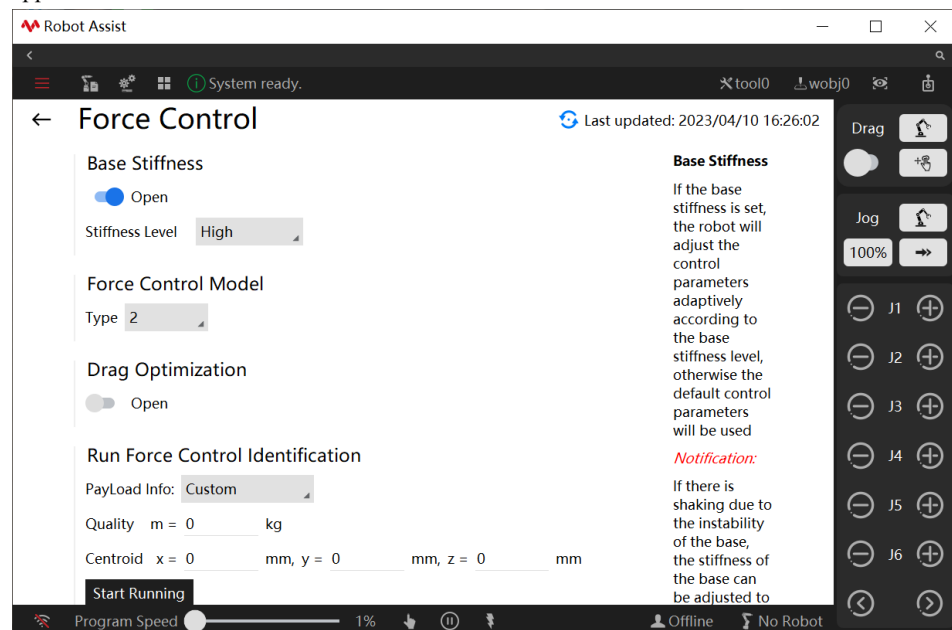
Base stiffness

There are two base stiffness modes: high and low. When the base stiffness level that matches the actual installation environment is set, the robot will switch the corresponding basic control parameters.

The high base stiffness control parameters are set to default after initialization and when the function is disabled.

Base Stiffness

🔵 Open

Stiffness Level | High ▾ |

The parameter does not require adjustment for common base scenarios. When the robot is mounted on a flexible base or mobile platform, the base stiffness level needs to be changed to low. For example, the robot sits on an AGV trolley.

Base Stiffness

🔵 Open

Stiffness Level | Low ▾ |

Force control model

The force control model enables the configuration of basic force control parameters. Be careful with this developer option.

The 0 model is set to default after initialization and when no modification is made.

Force Control Model

Type | 0 ▾ |

There is no need to adjust this parameter when the robot operates properly.

Drag optimization

Drag optimization is to improve the dragging experience and better force control in extreme conditions.

The main purpose is motor overcurrent protection when the drag ends. The strategy of slowing down and stopping is adopted for a better dragging experience. This function is enabled by default after the initialization and when no modification is made.

Drag Optimization

🔵 Open

Force control parameter identification

It refers to the accurate identification of the control parameters by adding external loads.

Run Force Control Identification

PayLoad Info: | Custom ▾ |

Quality   m = 0   kg

Centroid   x = 0   mm, y = 0   mm, z = 0   mm

Start Running

> ⚠️ Warning
>
> This function is for the experienced one only. Guidance by professionals or reference to the user manual is required.
>
> 1. Dedicated calibration blocks should be first used as the loads in the identification process.
> 2. The actual load can also be input by customization when there are no appropriate tools.

7.1.9 Quick turn settings

Explanation

The HMI interface offers quick turn functions to conveniently adjust the robot to common poses. Users can customize common poses and turn to a new custom orientation quickly on the HMI interface after the setting is completed. It supports custom poses including the drag pose, shipping pose, and Home pose etc.

This function can also turn the robot to some special orientations quickly while keeping the TCP position and elbow (only available for 7-axis robots) unchanged, including the flange parallel to the ground, the X axis of the tool frame perpendicular to the ground, the Y axis of the tool frame perpendicular to the ground, and the Z axis of the tool frame perpendicular to the ground.

Operation

The quick pose adjustment is available in Manual Mode in a way similar to Jog operation. In Manual Mode, the robot is powered on via the enable device. When the button for the corresponding target pose is pressed, the robot will move to the target pose in the joint space.

The motion speed can be adjusted via the Jog speed.

Parameter configuration

The Quick Turn features parameter configuration. For users who want to use other shipping poses, drag poses, or Home poses, they can set the parameters in the Robot -> Quick Turn page.

Turn on the Enable button, click the corresponding Quick Turn button in the Motion window, and the robot will move to the modified position. If the parameter configuration is not enabled, Quick Turn adopts the default pose.

Home pose

The following describes the settings of the Home pose and the definition of parameters in detail. The Home pose can be set to a range based on the joint angle. When the robot joints remain within this range, it is regarded that the robot is at the Home pose, and the system IO "Home State" is output. The reference point of the Home pose can be taught and updated with "the current pose".

The following are the parameters:

Home Pose				
○ Default ● Custom				
J1	0	home err	0.1	+/- (0.1 - 30)
J2	0	home err	0.1	+/- (0.1 - 30)
J3	0	home err	0.1	+/- (0.1 - 30)
J4	0	home err	0.1	+/- (0.1 - 30)
J5	0	home err	0.1	+/- (0.1 - 30)
J6	0	home err	0.1	+/- (0.1 - 30)

Move To   Refresh Pos

No.	Name	Meaning
1	Reference Value	The reference value of origin for each joint.
2	Offset	The float value of the origin range symmetrically around the reference value. Offset value range: [0.1,30]. For example, if the reference value is 1° and the offset value is 3°, the origin falls in the range of [-2°,4°].

The relationship between the origin range, the reference value and the offset value is shown as follows:
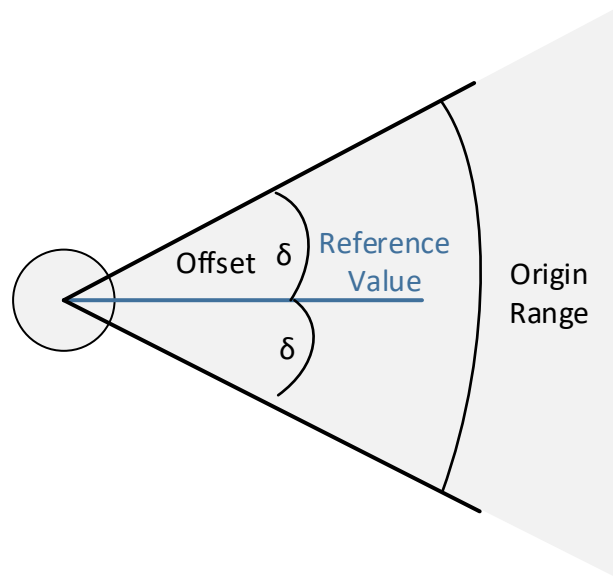
7.1.10 Electronic nameplate

Explanation

The electronic nameplate designed for industrial robots is installed in the robot body. It is mainly used to save the data of the robot body and avoid the loss of basic data after the replacement of the industrial computer or the controller cabinet.

The software functions of the electronics nameplate are mainly performed by the controller and Robot Assist. The controller performs data reading, verification, overwriting, etc., while Robot Assist is used to send operation commands related to the electronics nameplate and display data. After the controller is turned on, it will first check if there is an electronic nameplate. If there is an electronic nameplate, it will read the data normally, perform data verification, and store the verification result; If there is no electronic nameplate and the user does not choose to use the electronic nameplate, it will directly operate with the controller data; If there is no electronic nameplate and the user chooses to use the electronic nameplate, a prompt "there is no electronic nameplate" will appear. After Robot Assist is connected to the controller, it will first check the verification results of the electronics nameplate data in the controller, and give different pop-up prompts based on the verification results.

Users can simply follow the pop-up prompts. For details, refer to Chapter 1.1.2.2.

Startup selection

After startup, the controller checks if the data in the electronic nameplate is the same as that in the controller;

If they are same, the data in the controller will be used directly without any prompt;

Otherwise, a pop-up window will prompt whether to use the data in the electronic nameplate, as shown in the figure below;
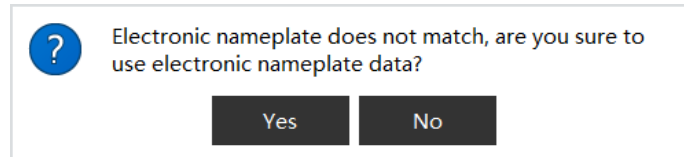
If the data in the electronic nameplate is successfully used, it will overwrite the data in the controller by default.

Here are several situations in which pop-up prompts appear:

1) If an electronic nameplate is detected and its data is different from that in the controller, a pop-up window will prompt "Do you want to use the data in the electronic nameplate?". Select "Yes" to use the data in the electronic nameplate and "No" to use the data in the controller directly;

2) After choosing to use the data in the electronic nameplate once, the electronic nameplate data will be used by default after restart. If the data in the controller is again different from that in the electronic nameplate, a pop-up window will prompt, "Do you want to use the data in the electronic nameplate?";

3) If the electronic nameplate is not detected during startup, the controller data will be used by default. If the electronic nameplate data is used once and cannot be detected after restart, a pop-up window will prompt "Do you want to use the data in the controller?". Select "Yes", the controller data will be used normally. Select "No", the controller will be in a malfunction state and cannot be operated. In this case, restart the controller to solve the problem.

 Notes

1. When the model data in the electronic nameplate does not match that in the controller, the data in the electronic nameplate cannot be used. To use the electronic nameplate data successfully, please ensure the model data in the controller is same as that in the electronic nameplate.
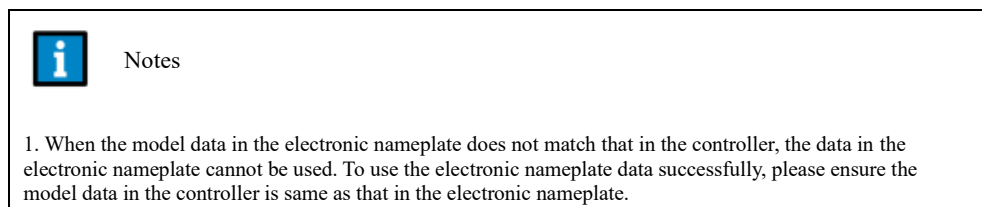
Electronic nameplate interface

Click Robot Configuration -> Settings -> Electronic Nameplate on the Robot Assist interface to display the information about the electronic nameplate. If an electronic nameplate is detected by the controller, regardless of whether the electronic nameplate is used, the information of the electronic nameplate parameter segments will be displayed on the interface;

The status of the electronic nameplate can be determined by the status bar on the interface with three parameters: the electronic nameplate status, whether the electronic nameplate data matches the controller data, and whether the Use Electronic Nameplate button is pressed during startup, as shown in the figure below:

If an electronic nameplate is not detected during startup, the interface is as shown in the figure below:

Nameplate

Status: Not Existed & Unused

Export RC

ℹ️ Notes

1. Regarding the third parameter in the status bar, if the Use Electronic Nameplate button is pressed, the parameter displays In Use regardless of whether the data is successfully used.

Functions of electronic nameplate interface

Function	Description
Export controller data	Export the data of relevant parameter segments in the controller to a file
Export electronic nameplate data	Export the data in the electronic nameplate to a file
Refresh	Synchronize the information of the electronic nameplate
Basic information	The parameter segments about the basic information of the electronic nameplate. It is unable to be modified manually
Encoder battery voltage	The actual battery voltage of the encoder. It is measured during startup and every 24 hours after a startup. It is unable to be modified manually
Run time	When the motor runs, the run time increases accordingly. The value is refreshed every hour on the interface. This parameter cannot be modified manually;
Mechanical zero parameters and kinematic parameters	The current values of the controller and the electronic nameplate will be displayed on the interface, respectively. This parameter cannot be modified manually;
Dynamic parameters	The parameter segment is not displayed on the interface;
Overwrite electronic nameplate data	Overwrite the data in the electronic nameplate with the data in the controller.

Nameplate

Status: Normal & UnMatch & Used

Export RC   Export Nameplate   Refresh

Basic Information

ID 18446744073709552000   Robot Model NB4_R475_04H7   Hardware Version AAA_0803   SN AAA   Data Format 4294967295

Monitor Information

Battery Voltage(v) 0   Running Time(h) 0

Mech Zero

Robot Controller

Calibrate Time   2022-05-27 16:16:22

Encoder Multiple   [-1, 6, -7, 0, 11, -22]

Encoder Single   [-61814, 87900, -19347, -22489, 24441, -9069]

Nameplate

Calibrate Time   1970-01-01 08:00:00

Encoder Multiple   [-1, -1, -1, -1, -1, -1]

Encoder Single   [-1, -1, -1, -1, -1, -1]

Cover Nameplate

Kinematics Para

Robot Controller

Calibrate Time   2022-05-27 16:17:49

Robot Dimension [0, 0, 0, 30, 0, 380, 0, 0, 440, 109.5, 0, 35, 325.5, 0, 0, 83, 0, 0, 0, 0]

Nameplate

Calibrate Time   1970-01-01 08:00:00

Robot Dimension [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Cover Nameplate

Dynamics Para

Cover Nameplate

ℹ️ Notes

1. All exported data are encrypted.

2. When the electronic nameplate is used, the controller automatically synchronizes the modified data to the electronic nameplate after the robot performs zero calibration, robot parameter modification, or dynamic parameter identification.

7.2 Safety Features

7.2.1 Scope

Safety Features	Industrial Robot	xMate cobot
Soft limit	Y	Y
Virtual wall	N	Y
Collision detection	Y	Y
Safety area	N	Y
Safety monitor	N	Y
Collaboration mode	N	Y

7.2.2 Soft limit

Function Description

Soft limit is the function that sets the maximum motion range of each axis at the software level. Users can set the soft limit according to the site conditions to avoid interference or collision between the robot and peripheral equipment. The following figure takes a seven-axis robot as an example. The number of axes and the soft limit of each axis vary with the model.
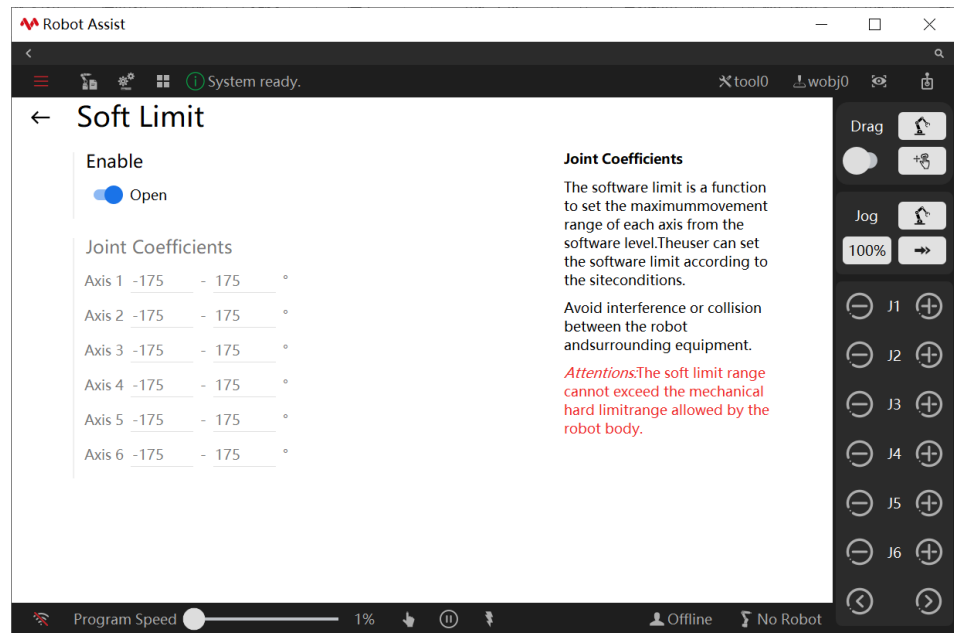
> ⚠️ **Warning**
>
> The range of soft limits cannot exceed the mechanical hard limit range allowed by the robot body.

When the robot is beyond the soft limit

In some rare cases, the robot may move beyond the soft limit. For example, if the robot triggers an emergency stop when it reaches the limit boundary,

it may exceed the soft limit when executing STOP0. If the robot has one or more joints beyond the soft limit, Jog and running programs cannot be performed. At this time, the soft limit must be canceled first, then return the overrun joint jog to the range within soft limit and enable again the soft limit.

> ⚠️ **Warning**
>
> Cancellation of the soft limit function can only be used to Jog the overrun joint back to the normal range when the robot joint exceeds the soft limit.
> The program will not run when the soft limit is canceled.

7.2.3 Virtual wall

Function Description

xMate cobot provides virtual walls targeting certain medical care scenarios. For example, if xMate is used as a physician's aid, the user can drag it to perform surgical operations. Virtual walls can be set through the surgical navigation system to limit the operating space of the xMate end-effector flange.

➢ Virtual walls can be cuboids or spheres;
➢ The center and limit range of the virtual walls can be set as below:
  ➢ The center (unit: mm) is set with the base frame as the reference frame, and the current flange position can be set as the center of the virtual walls;
  ➢ The limit range of the spherical virtual walls is defined by the spherical radius (unit: mm);
  ➢ The limit range of the cuboid virtual walls is defined by length (X), width (Y), and height

(Z) (unit: mm);
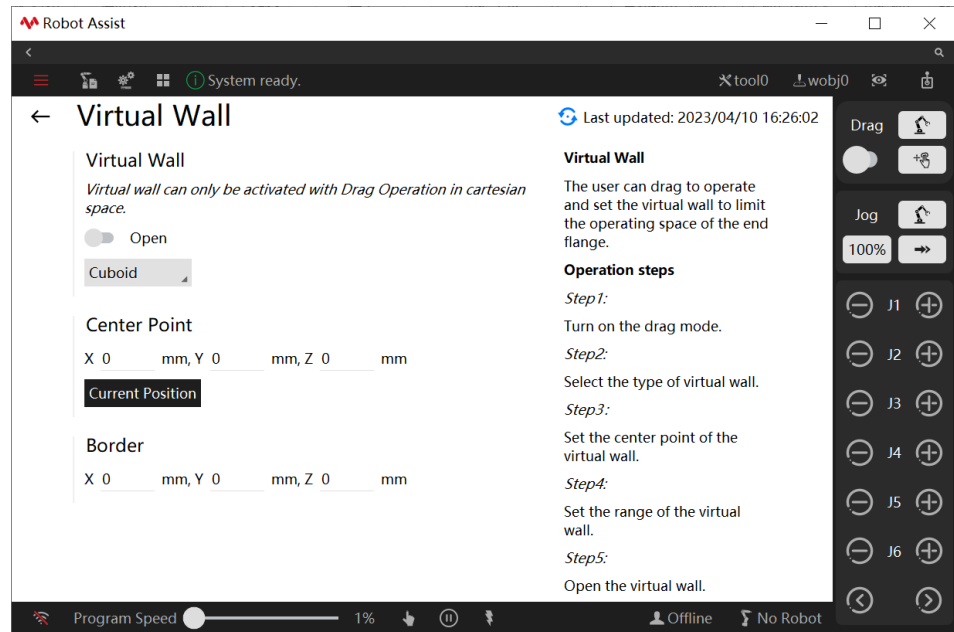
How to set:

	Operation	Description
1	Use admin to log in to the system and activate the drag mode.	Virtual walls are only valid when the drag mode is on.
2	Select the virtual wall shape type.	Cuboid and sphere are supported.
3	Determine the center of the virtual walls.	Drag the robot to a location and click the Current Location button on the interface to set the center of the robot flange as the virtual walls.
4	Set the range of the virtual walls.	The limit range of the spherical virtual walls is defined by the spherical radius (unit: mm). The limit range of the cuboid virtual walls is defined by length (X), width (Y), and height (Z) (unit: mm).
5	Enable the virtual walls.	Click the Open button to activate the virtual walls.

7.2.4 Collision detection

Function Description

The collision detection function is a passive detection based on the robot's dynamics model parameters. It detects a collision and implements preset countermeasures when the robot collides with the outside.

➢ Collision detection is disabled by default.

➢ The collision detection mode contains the level setting and the single-axis setting.

Level setting: For different application scenarios, three levels of detection sensitivity are available: low, medium, and high. The higher the sensitivity, the smaller the external force that triggers collision detection. Low sensitivity is suitable for full-load full-speed, medium sensitivity for half-load 50% automatic operation, and high sensitivity for JOG or collaboration mode. For example, high sensitivity can be selected when the user jogs the robot and wants to activate collision detection. Low sensitivity is recommended when the program is running at full speed with a full load in Automatic mode.

Single-axis setting: It provides specific application scenarios for fine-tuning the interfaces of detection sensitivity. The user can adjust the sensitivity axis by axis according to the collision information provided on the HMI. It allows the user to set the sensitivity suitable for the current application scenario while balancing between sensitivity and stability.

➢ Collision detection provides two trigger behaviors, action pause and safety stop.

Action pause: When detecting a collision, the robot will pause its current motion. When applied with a downward force, the robot will resume its motion;

Safety Stop: When detecting a collision, the robot stops the current motion safely.
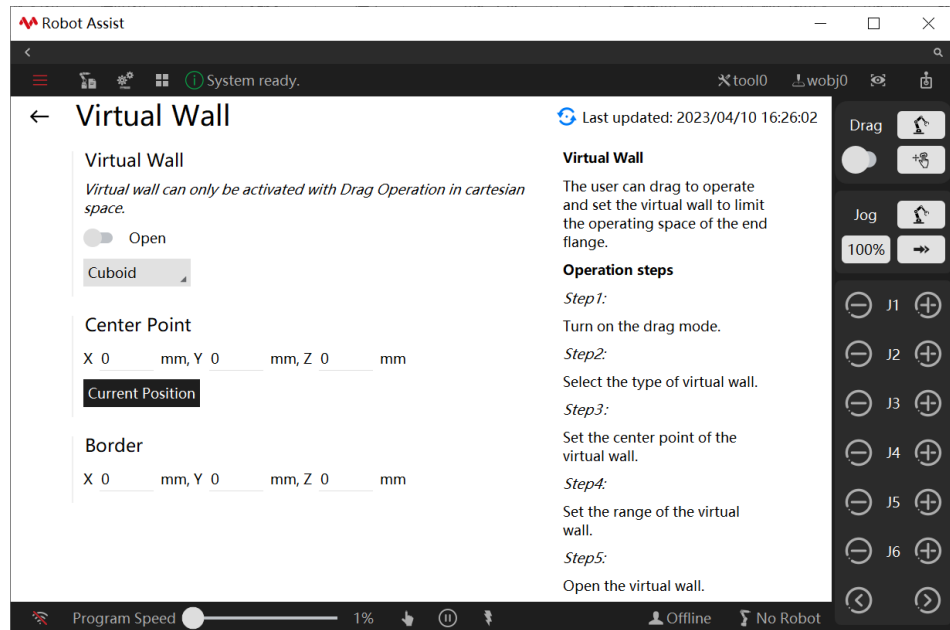
Notes

1. During the execution of the program, when the robot moves at a high speed and collides with external devices (stiff collision), and the collision force is too high, causing the servo driver to alarm and stop, the robot can only run again when the collision is cleared, the robot restarted, and the servo alarm reset.

2. Incorrect sensitivity mode selected may cause false collision alarm. Please select different sensitivity thresholds for each application scenario.

3. The collision detection sensitivity is affected by the robot hardware, and there are differences in sensitivity thresholds between different robots. Currently, the three sensitivity modes only provide a set of nominal values. Users with have higher requirements for collision detection sensitivity can fine-tune the sensitivity of each axis based on specific application scenarios through the single-axis setting or adjust the detection sensitivity online through RL commands.

> ⚠️ **Warning**
>
> Before using collision detection, the user must ensure that the following parameters are set correctly. Otherwise, the controller may fail to calculate the correct output torque, resulting in a false alarm.
>
> 1、 Robot model
> 2、 Robot installation method
> 3、 Load information (tool)
> 4、 Mechanical and sensor zeros
> 5、 Robot body parameters

7.2.5 Safety area

Function Description

Safety areas restrict the robot's motion space. The user can define a number of safety areas in the space (up to 10 in the controller system). When entering and exiting such safety areas, the robot can

selectively trigger the preset safety behaviors and automatically modify the register value of the corresponding register function code bound to the safety area.
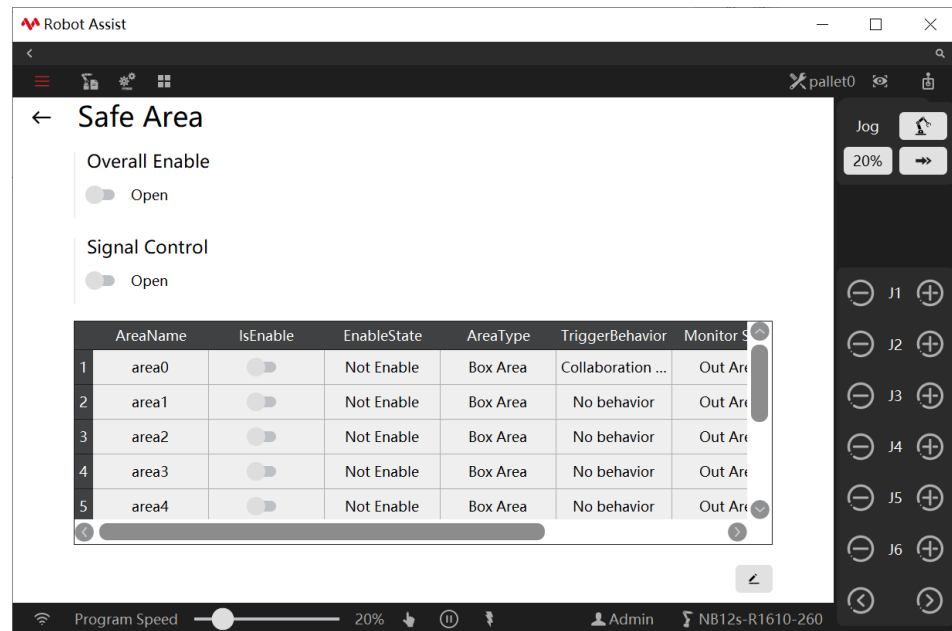
The safety area function can be turned on or off by the "Overall Enable". When this switch is off, all safety areas become invalid.

Each safety area can be turned on or off independently through HMI or register.

A safety area can be turned on or off by signal: when "signal control" is on, whether a safety region is "turned on or not" depends on the register bound with the function code "enable_safe_region01~enable_safe_region10" (type: bool or int16, read/write: read-only), and the button under the "IsEnable" column is disabled; when "signal control" is off, whether a safety region is "turned on or not" can be directly set through HMI.

Users can modify the properties of any safety area.

Parameter explanation	
	➤ Safety areas can be rectangular or point-plane vector;
	➤ Trigger behaviors include: no behavior, safety stop, and collaboration mode;
	■ No behavior: the robot has no specific action;
	■ Safety stop: the robot executes stop1 to stop and power off;
	■ Collaboration mode: the robot enters the collaboration mode (cobots only).
	➤ The state of the region-bound register after triggering can be set: true/false;
	■ True: the register output signal is true when the safety area triggers a safety behavior, and vice versa; (output true when entering the forbidden zone)
	■ False: the register output signal is false when the safety area triggers a safety behavior, and vice versa; (output false when entering the forbidden zone)

For the rectangular region, the pose of the region's center can be determined by the pendant (TCP relative to the robot base frame) or manual input; the length, width, and height of the cuboid can be set manually; the orientation of the cuboid can also be set: inside indicates that when TCP moves into the cuboid, a safety behavior and corresponding register state modification will be triggered, and the inside space of the cuboid is the forbidden region; outside indicates that when TCP moves out of the cuboid, a safety behavior and corresponding register state modification will be triggered, and the inside space of the cuboid is the safety area while the space outside the cuboid is the forbidden zone.
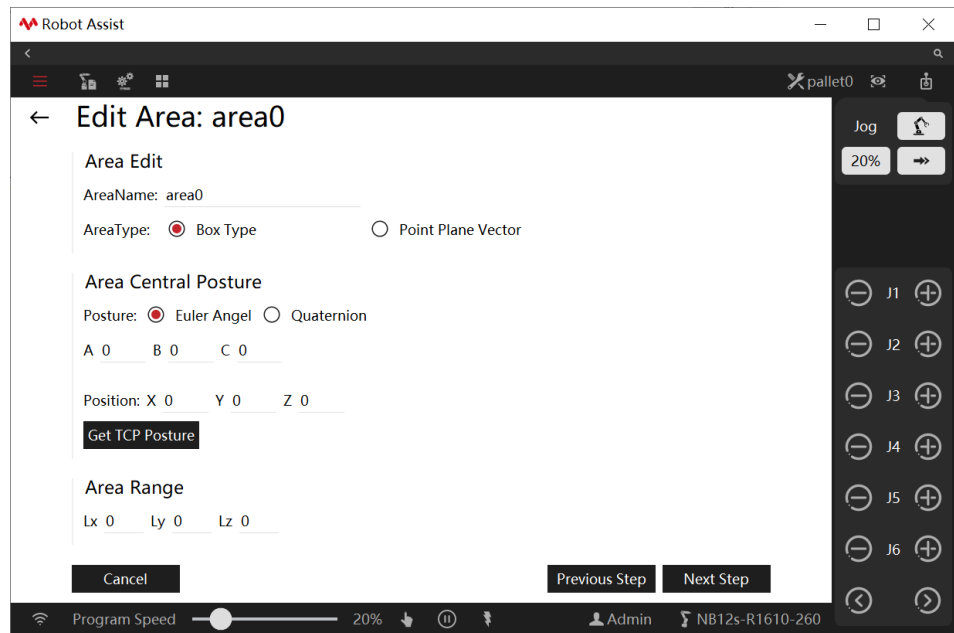
For the point-plane vector zone, the pose can only be set through manual input; the safety area is in the direction pointed by the Z-axis defining the orientation. Direction property: positive indicates that the robot TCP enters the safety area to trigger a safety behavior and corresponding register state modification; negative indicates that the robot TCP leaves the safety area to trigger a safety behavior and corresponding register state modification.
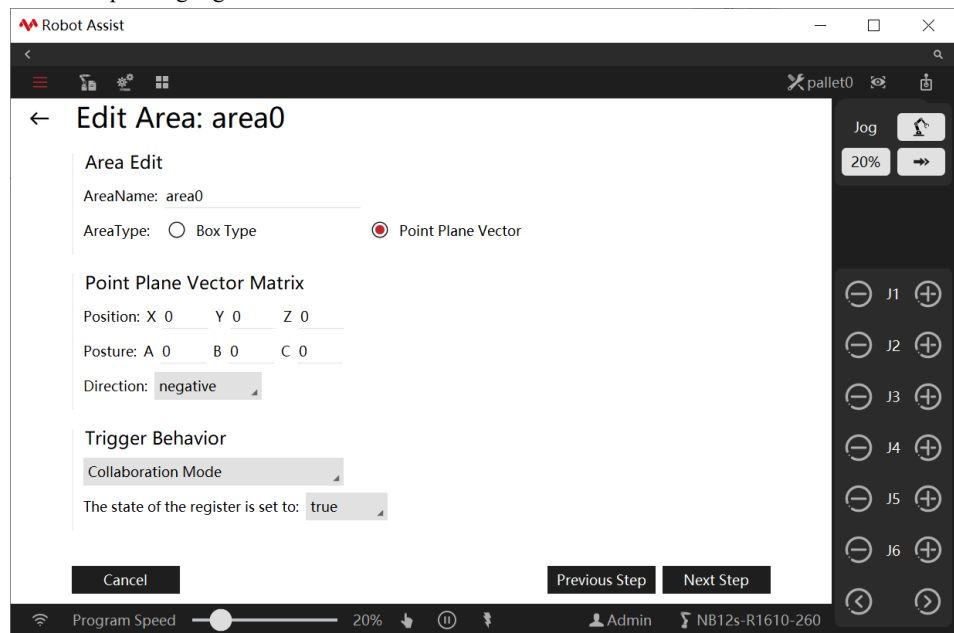
Function bound register

Up to 10 control switches of whether the safety area is in effect and the trigger state of the safety area can be bound to a register. The safety area switch is controlled externally, and the safety area trigger state is fed back to external devices.
To bind the trigger state of the safety area to a register, first create a new register as shown below. Select write only and "sta_safe_region01~sta_safe_region10", which means the trigger state of the corresponding safety area will be bound to the new register.
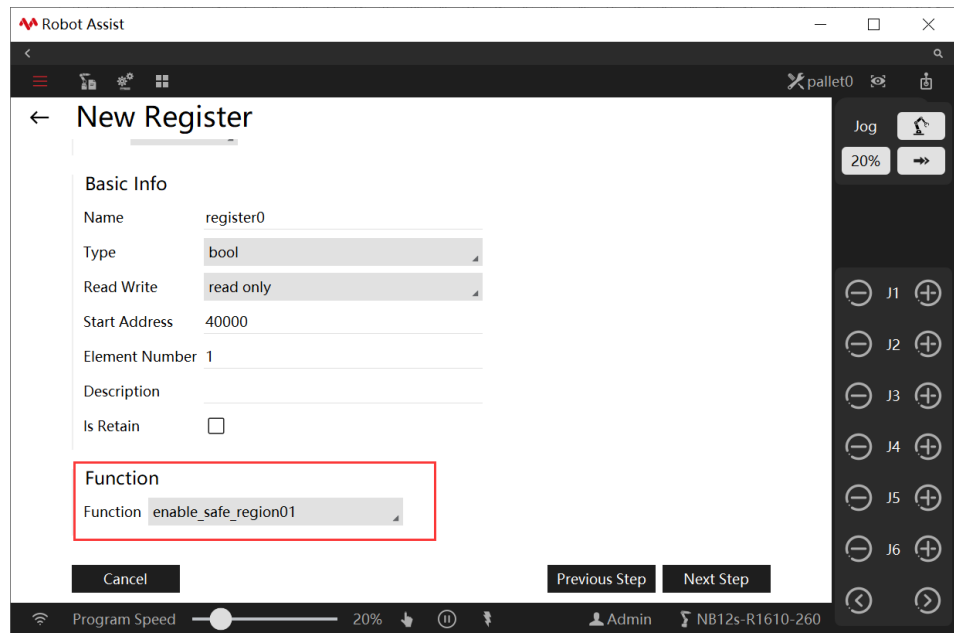
To bind the power switch of the safety area to a register, first create a new register as shown below. Select read only and "enable_safe_region01~enable_safe_region10", which means the control switch of the corresponding safety area will be bound to the new register.

7.2.6 Safety monitor

Explanation

Safety monitor targets the robot's normal operation. In this mode, thresholds of monitoring items are higher than those in the Collaboration mode.

Parameter configuration

Admin or higher permission is required to configure safety monitoring as shown in the below interface:

1. Setting all monitoring items and parameters in the Safety Monitoring interface, including:

A	Maximum speed of each joint, with each axis set independently; Maximum speed of each joint: [180, 150, 180, 180, 225, 225, 225] °/s
B	Maximum linear speed of TCP: 1m/s, the threshold is shared by X/Y/Z with an available speed range of 0.0 - 1 m/s
C	Maximum torque of each joint, with each axis set independently; Maximum torque of each joint for xMate3 Pro: [281.7, 338.1, 281.7, 281.7, 99.6, 99.6, 99.6] Nm. Maximum torque of each joint for xMate7 Pro: [720, 720, 281.7, 281.7, 124.5, 124.5, 124.5] Nm.
D	Total power limit: 4476W. Available power range: 0 - 4476W

2. When monitoring items are triggered, the robot performs STOP1.

3. Each monitoring item can be enabled/disabled independently and are off by default.
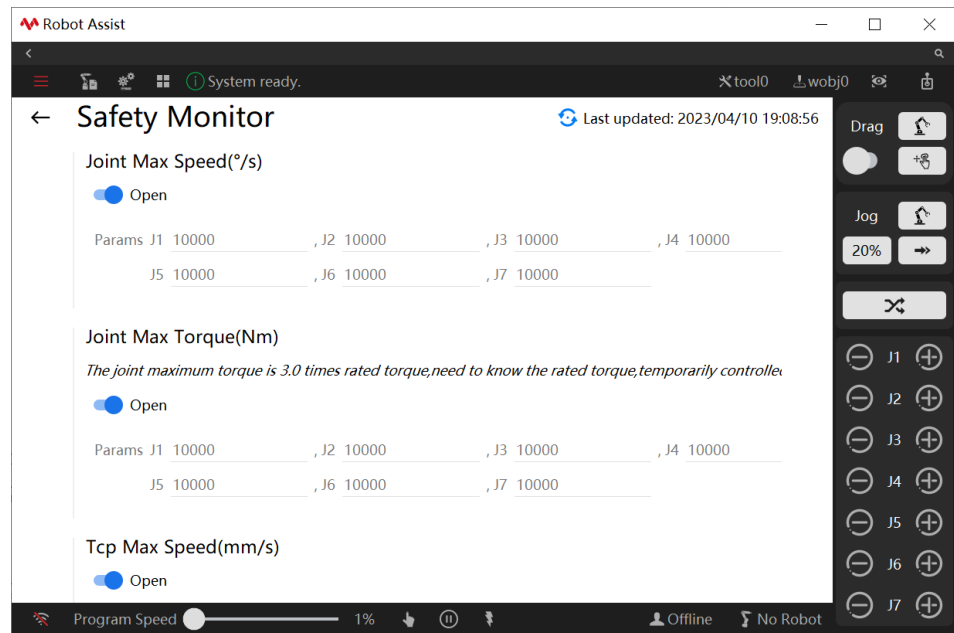
7.2.7 Collaboration mode

Explanation

The Collaboration mode is a working mode where humans and robots share the working area. In this mode, the robot running speed will be reduced based on the TCP maximum speed monitoring parameter setting in the Collaboration mode.

Parameter configuration

Admin or higher permission is required to configure Collaboration mode as shown in the below interface:
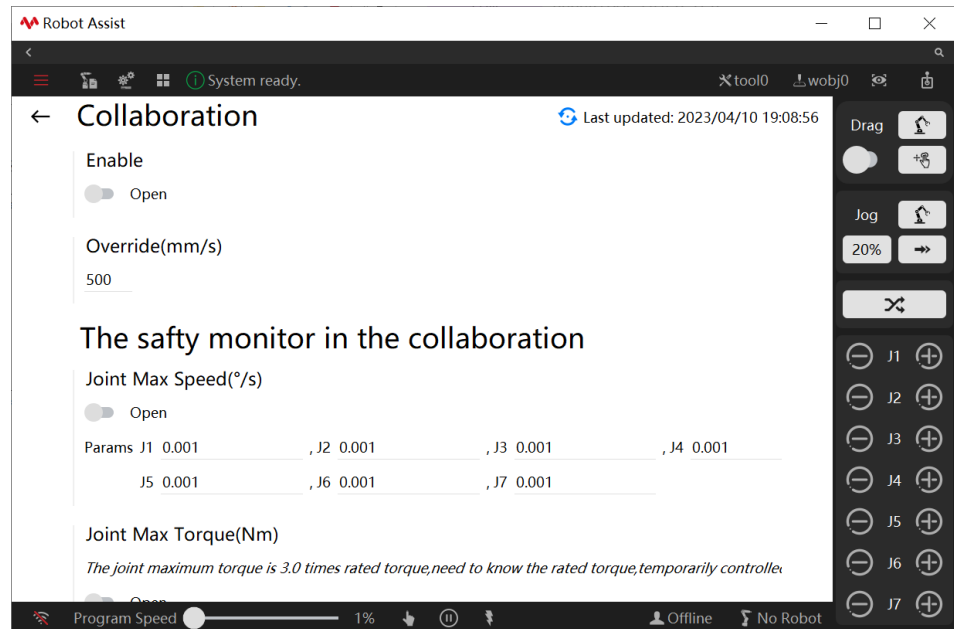
1. The range of monitoring parameters set in the Collaboration mode is as follows:

A	Maximum joint speed: 15°/s. Each axis is set independently with the range of 0.0~15.0°/s
B	Maximum linear speed of TCP: 0.25 m/s, the threshold is shared by X/Y/Z with an available speed range of 0.0 - 0.25 m/s
C	Maximum torque of each joint, with each axis set independently; Maximum joint torque: [140.9, 169.0, 140.9, 140.9, 49.8, 49.8, 49.8] Nm.

D	Total power limit: 192.7 W. Available power range: 0 - 192.7 W

2. In the Collaboration mode, STOP1 emergency stop is triggered when monitoring parameters exceed limits.

7.2.8 Safety position

Explanation

xCore control system supports up to 8 safety positions with joint angles as reference. Each safety position corresponds to a register function code (type: bool or int16, read/write: write only, sta_safe_jnt_pos1~sta_safe_jnt_pos8). When the current joint angle of the robot and the joint angle set for a safety position are within the allowable error, the value of the register to which the corresponding register function code for the safety position is bound to will be modified automatically (when within the allowable error of the safety position, if the register type is bool, the register value is true; if the register type is int16, the register value is 1). The user can understand the robot's position relative to the safety position through this function.

Parameter configuration
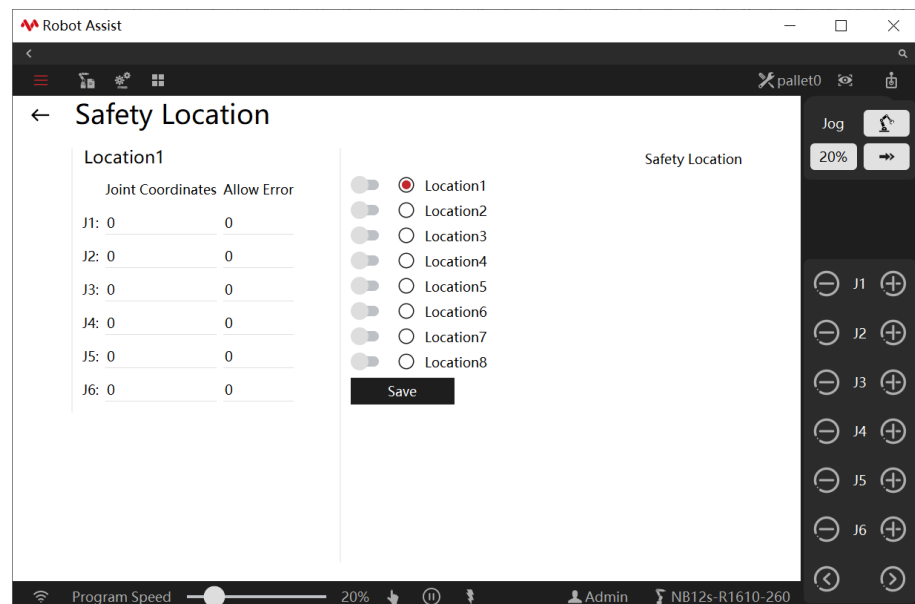
On the above page, click to select a safety position, move the robot to the desired position, and manually set the allowable error of each joint; click "Save" to record the current safety position to complete the setting.

The user can set whether to enable any safety position.

Function bound register

Up to 8 safety position states can be bound to a register, and the trigger state of whether reaching the safety position is delivered to external devices through the register signal.

To bind the safety area state to a register, first create a new register as shown below. Select write only and "sta_safe_jnt_pos1~sta_safe_jnt_pos 8", which means the feedback state of the corresponding safety position will be bound to the new register.
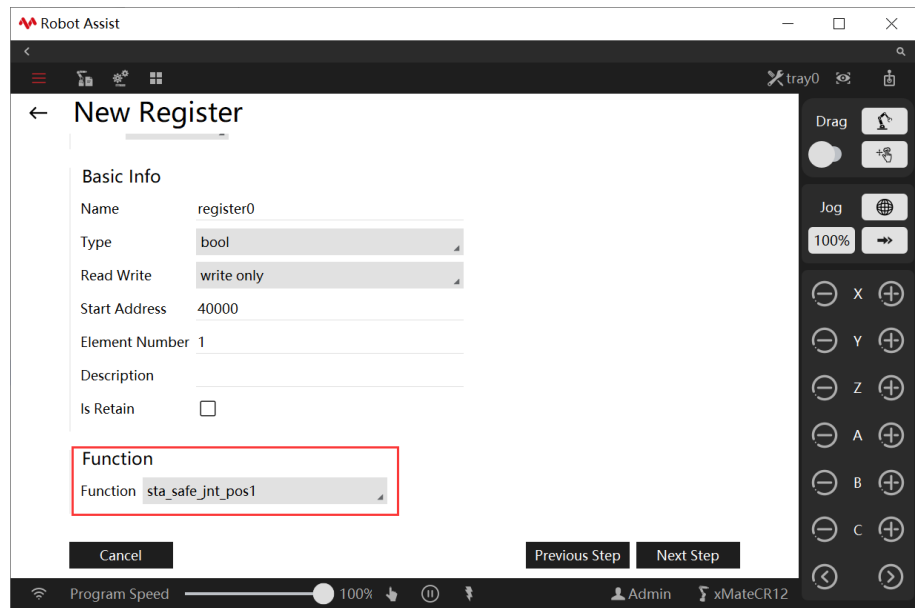
7.3 Communication Configuration

7.3.1 System IO Configuration

Explanation

The system IO is divided into system digital input and system digital output. The external controller can send various commands to the xCore control system through system input, such as motor power-on, startup procedure, emergency stop reset, etc. Also, the xCore system can send various states using the system output IO.

System input

The system inputs supported by the xCore system include:

No.	System input	Remarks
1	Motor ON	
2	Motor OFF	
3	Program Start	
4	Program Pause	
5	PP to Main	
6	Enter Collaboration	Cobots only
7	Exit Collaboration	Cobots only
8	Clear Alarm	
9	MotorOn & Run	Power on, pptomain, run in order
10	MotorOn & Continue	Power on and run
11	MotorOff & Pause	Pause, wait for the robot to stop, and power off
12	Emergency & Clear Alarm	
13	Switch Manual	
14	Switch Auto	Only the function is effective in the manual mode
15	Open Drag	Only for cobots. Need to enable Drag mode on the interface
16	Close Drag	Only for cobots. Need to enable Drag mode on the interface

All system inputs are pulse-triggered. To ensure that the xCore system receives external commands

correctly, please ensure that the pulse width of the external input is not less than 300 milliseconds.

> **ℹ Notes**
>
> The system input function is only valid in Automatic mode, and the signal from the system input in manual mode will be ignored.

System output

The system outputs supported by the xCore system include:

No.	System output	Valid output	Invalid output	Remarks
1	Motor State	Motor power-on	Motor power-off	
2	Running State	Program running	Program not running	
3	Operate Mode	Automatic mode	Manual mode/Wait mode	
4	Estop State	Emergency stop	Non emergency stop	
5	Collision Detection	Triggered	Not triggered	Cobots only
6	Collaboration State	Collaboration mode	Non-Collaboration mode	Cobots only
7	Alarm State	Alarm	No alarm	
8	Home State	The robot TCP is at Home	The robot TCP is not at Home	

All other system output signals are active at a high level except the "Operating Mode" signal.

For the signal "Operating Mode", the output is at a high level in Automatic mode and low in Manual mode.

> **ℹ Notes**
>
> The system output status is valid in both manual and automatic modes. However, for safety and availability considerations, these signals are only to be used when the xCore is in Automatic Mode.

Use restrictions

After an IO point is bound to the system IO, it cannot be forced to output or simulate input operations.

7.3.2 External communication

Explanation

The xCore system provides a Socket-based external communication interface through which host systems (PLC, MES, etc.) can send control commands to the robot or obtain the robot status.

The Socket communication interface supports the configuration of IP address, port number, and communication terminator (suffix).

The Socket communication interface supports the robot to serve as a client or server, but only one state at a time.

Enable interface

Before using the interactive commands, configure the parameters related to the Socket communication and enable the function. This is operated on the Teach Pendant interface. Go to the interface via Robot -> Communication -> External Communication, as shown in the figure below:
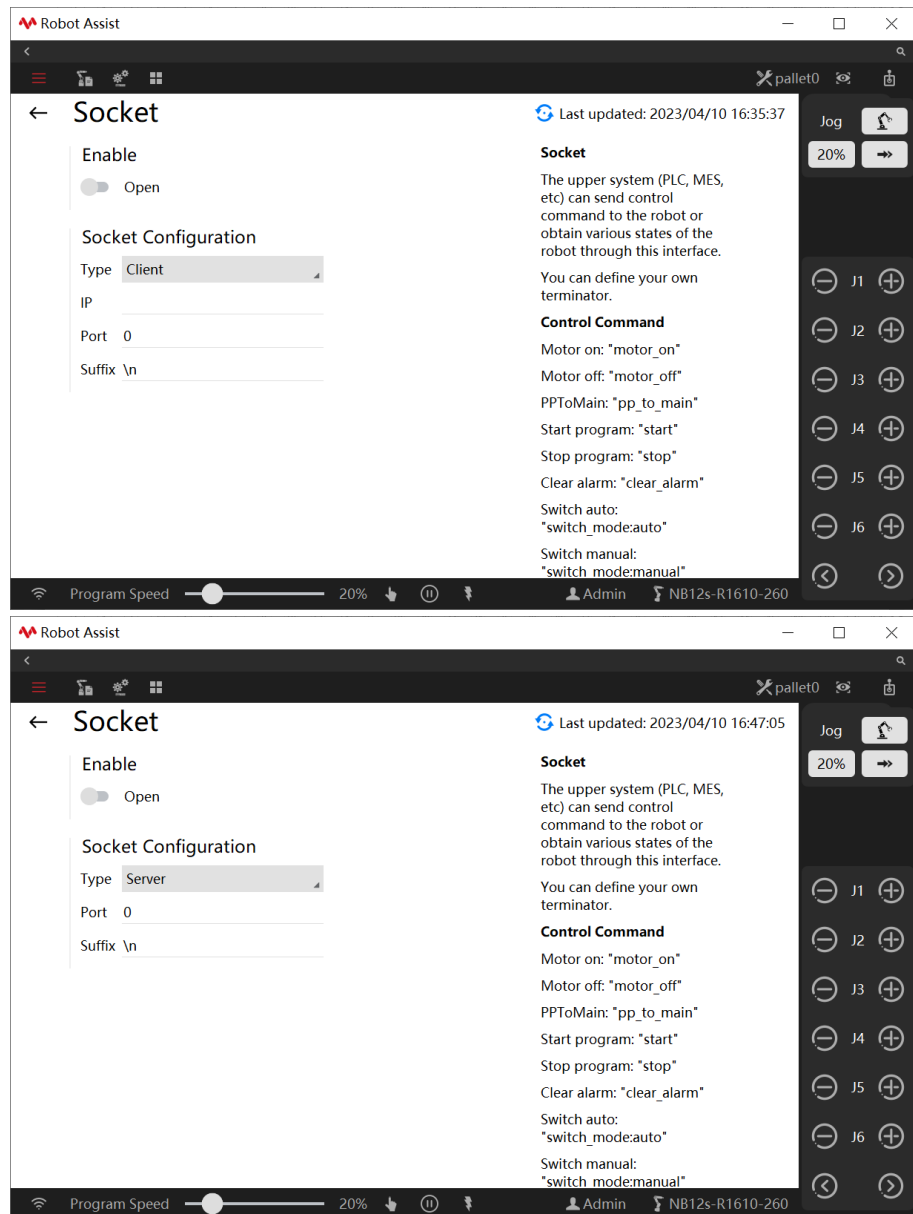
Parameters

When the robot is used as a client, the following parameters need to be configured:

No.	Parameters	Description
1	IP	Server IP, such as the IP address of the connected PLM and MES systems.
2	Port	Server-side listening port
3	Suffix	The suffix is the characters added to the end of the control commands or monitoring commands sent from the server to the robot. They are typically simple terminators such as \r, \n or \t. Please note combined suffixes can be used here without limitation on length, such as \r\n, \r\t or \r\n\t. Visible characters such as letters can also be used.

The robot used as a server supports multiple connections. In this case, please pay attention to the control sequence on the client side to avoid any conflict. The following parameters need to be configured:

No.	Parameters	Description
1	Port	Server-side listening port
2	Suffix	The suffix is the characters added to the end of the control commands or monitoring commands sent from the server to the robot. They are typically simple terminators such as \r, \n or \t. Please note combined suffixes can be used here without limitation on length, such as \r\n, \r\t or \r\n\t. Visible characters such as letters can also be used.

List of interactive commands

The following table shows the information content supported by the external communication interface and the corresponding command formats. Assume the user uses "\r" as the specified command terminator ("\r" is an escape character for carriage return, the decimal value is 13). Interaction commands include control commands, configuration commands, and monitoring commands.

Control commands include:

	Command name	String sent	Return value	Remarks
1	Close the socket interface	"xCore::SocketInterface::Disable" +"\r"	No return value	
2	Start the socket interface	"xCore::SocketInterface::Enable" +"\r"	No return value	
3	Start program	"start"+"\r"	"true" if success; "false" if failed	
4	Stop program	"stop"+"\r"	"true" if success; "false" if failed	
5	Clear servo alarms	"clear_alarm"+"\r"	"true" if success; "false" if failed	
6	Program pointer to main	"pp_to_main"+"\r"	"true" if success; "false" if failed	
7	Motor power-on	"motor_on"+"\r"	"true" if success; "false" if failed	
8	Motor power-off	"motor_off" + "\r"	"true" if success; "false" if failed	
9	Switch to Manual mode	"switch_mode:manual"+"\r"	"true" if success; "false" if failed	
10	Switch to Automatic mode	"switch_mode:auto"+"\r"	"true" if success; "false" if failed	
11	Enable Drag mode	"open_drag"+"\r"	"true" if success; "false" if failed	Cobots only
12	Disable Drag mode	"close_drag"+"\r"	"true" if success; "false" if failed	Cobots only

Monitoring commands shall include:

	Command name	String sent	Return value	Remarks
1	Motor power status	"motor_on_state" + "\r"	"true" if success; "false" if failed true: motor power on; false: motor power off	
2	Program status	"robot_running_state" + "\r"	"true" if success; "false" if failed true: running; false: not running	
3	Emergency stop	"estop_state" + "\r"	"true" if success; "false" if failed true: emergency stop; false: non emergency stop	
4	Fault	"fault_state" + "\r"	"true" if success; "false" if failed true: fault; false: not fault	
5	Operating mode	"operating_mode" + "\r"	"true" if success; "false" if failed true: Automatic mode; false: Manual mode/Wait mode	
6	Get Cartesian position	"cart_pos" + "\r"	Cartesian position string + "\r"	
7	Get Cartesian position	"cart_pos_name" + "\r"	"cart_pos: " + Cartesian position string + "\r"	
8	Get axis position	"jnt_pos" + "\r"	Axis position string + "\r"	
9	Get axis position	"jnt_pos_name" + "\r"	"jnt_pos: " + axis position string +"\r"	
10	Get axis velocity	"jnt_vel" + "\r"	Axis speed string + "\r"	Unit: rad/s
11	Get axis velocity	"jnt_vel_name" + "\r"	"jnt_vel: "+ axis speed string + "\r"	Unit: rad/s
12	Get axis torque	"jnt_trq" + "\r"	Axis torque string + "\r"	Unit: N.m
13	Get axis torque	"jnt_trq_name" + "\r"	"jnt_trq:" + axis torque string + "\r"	Unit: N.m
14	Home state output	"home_state" + "\r"	Returns "true" if there is an output or "false" if there is no output	

16	Collision detection state	"collision_state" + "\r"	Returns "true" if a collision detection is triggered or "false" if no collision detection is triggered	Cobots only
17	Obtain robot task state	"task_state" + "\r"	The task currently performed by the robot, including: ● Ready: ready ● Jog: jog ● Load identification: load_identify ● Dynamic Identification: dynamic_identify ● Enable Drag Mode: drag ● Program running: program ● Demo: demo ● RCI: rci ● Debug: debug	For details, please refer to the symbols and description of the robot's current status in Chapter 3.1.2 Bottom status bar.

Note:

	String format	Unit
Cartesian position	x, y, z, a, b, c, q1, q2, q3, q4	x, y, z, unit: mm; a, b, c unit: degree; q1~q4 are orientation quaternion
Axis position	j1, j2, j3, j4, j5, j6, j7	Robot axis degree, unit: rad; rail position, unit: m;
Axis velocity	vj1, vj2, vj3, vj4, vj5, vj6, vj7	Robot axis velocity, unit: rad; rail velocity, unit: m/s;
Axis torque	tj1, tj2, tj3, tj4, tj5, tj6, tj7	The unit of the robot axis and track torque is the thousandth of the rated torque of the motor;

7.3.3 Bus devices

Explanation

CC-Link, Modbus, EtherCAT, and PROFINET are supported. CC-Link includes CC-Link devices (adapted via EtherCAT) and CC-Link IE Field Basic. EtherCAT can be used to expand IO modules, PROFINET, EtherNet/IP, and other bus modules.

Supported Bus	Protocol	Supported method	Remarks
Modbus	TCP	Master and slave	
	UDP	Not supported	
	RTU	Master and slave	Industrial robots only
CC-Link	485	Remove device station (slave)	Industrial robots only
	IE Field Basic	Remove device station (slave)	
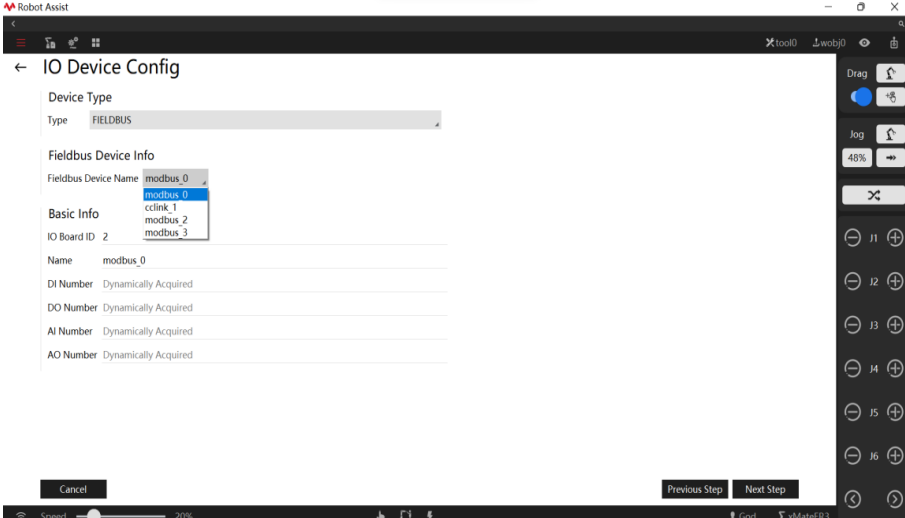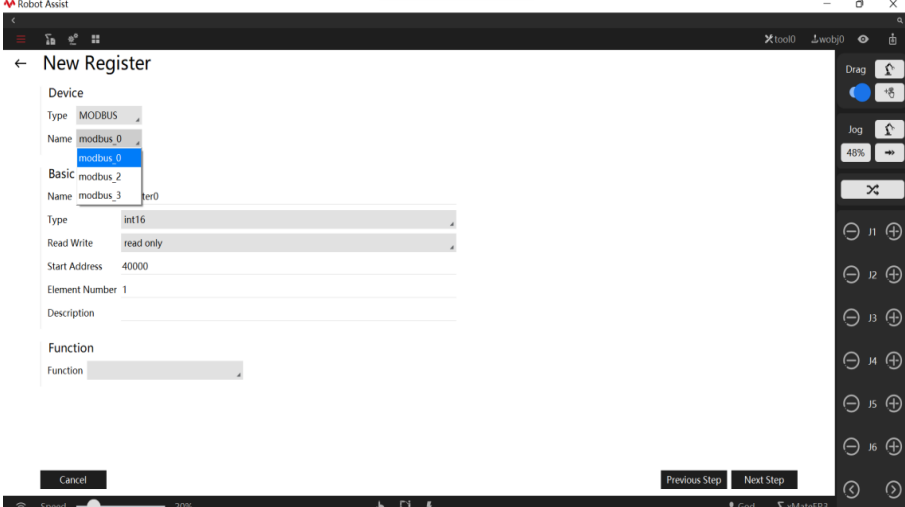
The following function codes are supported in Modbus:

Function code	Meaning	Supported
0x01	Read coil	Supported
0x05	Write a single coil	Supported
0x0F	Write multiple coils	Supported
0x02	Read discrete input	Supported
0x04	Read input register	Not supported
0x03	Read holding register	Supported
0x06	Write a single holding register	Supported
0x10	Write multiple holding registers	Supported

Configuration

Click Robot Configuration -> Communication -> Bus Device. The page is divided into two parts. All bus connections are managed on the upper part of the page, and the lower part displays the attribute parameters of a certain bus connection. On the upper part of the page, each bus connection can be enabled or disabled individually. When a bus connection is disabled, the IOs configured for the connection will not be displayed in Status Monitoring -> IO Signal.
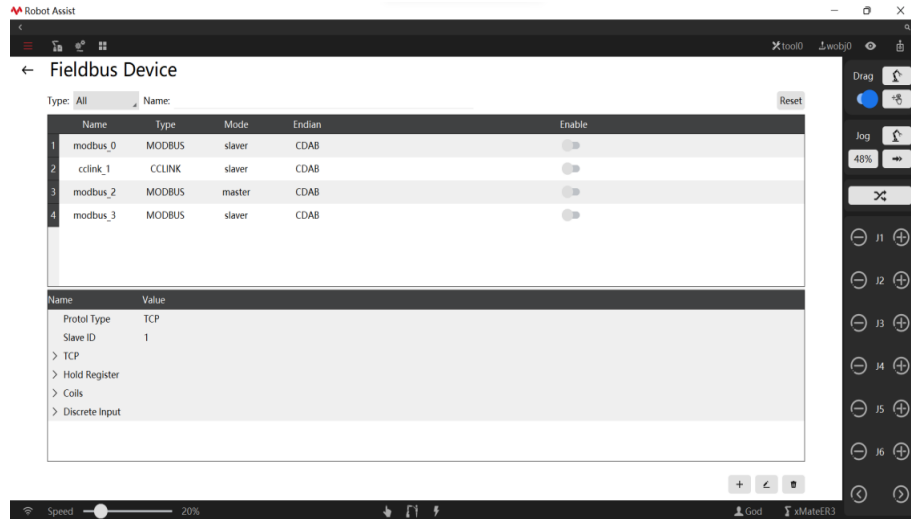
Parameter name	Parameter explanation
Name	The first column of the management list is the name of the bus connections, used in the IO Device and Register configuration. For example, the names in the above figure are modbus_0, modbus_1, modbus_2, and cclink, as shown in the figure below: ➢ This name field is used in IO device configuration to indicate which bus the IO device is related to; ➢ This name field is used in Register to indicate which bus the register is related to;
Type	The second column of the management list is the type of bus connections, which can be selected when adding/editing a bus

	device. Only CC-Link, Modbus, EtherCAT, and PROFINET are supported. CC-Link includes CC-Link devices (adapted via EtherCAT) and CC-Link IE Field Basic. EtherCAT can be used to expand IO modules, PROFINET, EtherNet/IP, and other bus modules.
Mode	The third column of the management list is the mode of the bus connections, indicating whether the current robot serves as a master or a slave on the bus.
Endianness	The fourth column of the management list is endianness, mainly used for registers. Since each register occupies 2 bytes, there are many hexadecimal sequences of the two bytes. This attribute needs to correspond to the master and the slave, otherwise, the data will not meet the expectations. Four types of endianness are supported in the control system: ABCD, CDAB (default), BADC, and DCBA.
Enabling button	The fifth column of the management list is the enabling button. It is used to enable or disable the bus function. Each bus device can be enabled or disabled individually. Please note that after a bus device is disabled, the IOs configured on the bus device will not be displayed in Status Monitoring -> IO Signal.

7.3.3.1 Modbus communication

Click at the lower right corner on the bus device page to enter the interface to add a new communication bus device, and select the device type MODBUS. It supports the TCP and RTU protocol, and the device can be configured as a master or slave.

7.3.3.1.1 Modbus TCP configuration

Parameter	Introduction
Mode	The robot can be selected as "master" or "slave".
Slave ID	When the robot serves as a slave, ensure that the overall configuration of the bus does not conflict with other slaves. When the robot serves as a master, it indicates the target slave ID that the robot expects to communicate with. Please note that when the robot serves as a master, it only supports single-slave communication with external devices;
TCP/IP	When the robot serves as a slave, fill in 0.0.0.0, which means all network cards are monitored. When the robot serves as a master, fill in the IP address of the target slave ID that the robot communicates with;
TCP port	The port number when the slave uses the TCP protocol;
Holding register start address	The start address of the register affected by the function codes 0x03, 0x06, and 0x10. Each register occupies 2 bytes;
Coil start address:	The start address of the register affected by the function codes 0x01, 0x05, and 0x0F;
Discrete input start address:	The start address of the register affected by the function code 0x02

7.3.3.1.2 Modbus RTU configuration
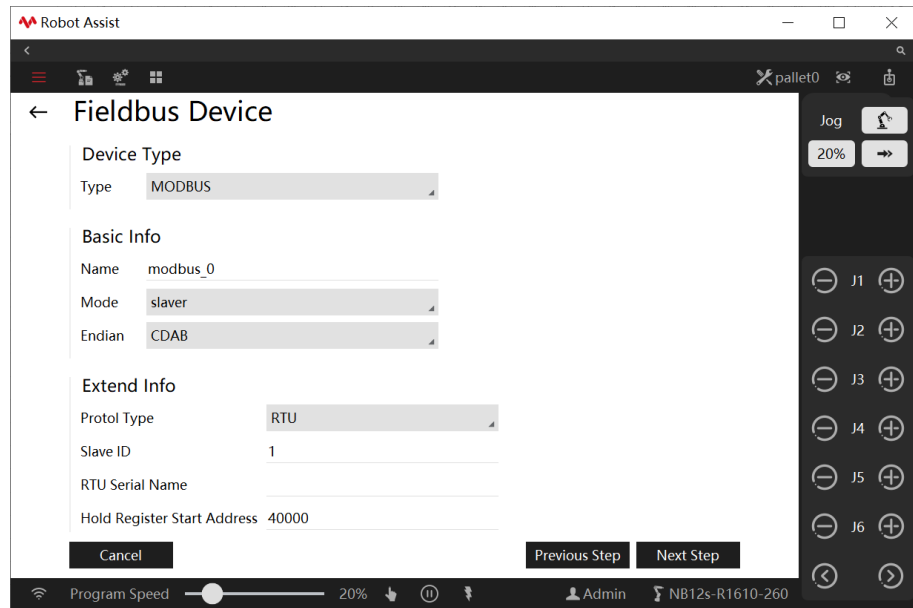
The Modbus RTU conception is partly the same as the Modbus TCP conception, which will not be repeated here. Only the differences are described as follows:

RTU serial port name: Indicates the serial port medium used for bus communication. Configure it in Robot Configuration -> Communication -> Serial Port Configuration, including the parameters for communication.

7.3.3.2 CC-Link communication

Click at the lower right corner on the bus device page to enter the interface to add a new communication bus device, and select the device type CCLINK. It supports the CC-Link and CC-Link IE Field Basic protocol, and the device can be configured as slave only.

7.3.3.2.1 CC-Link configuration
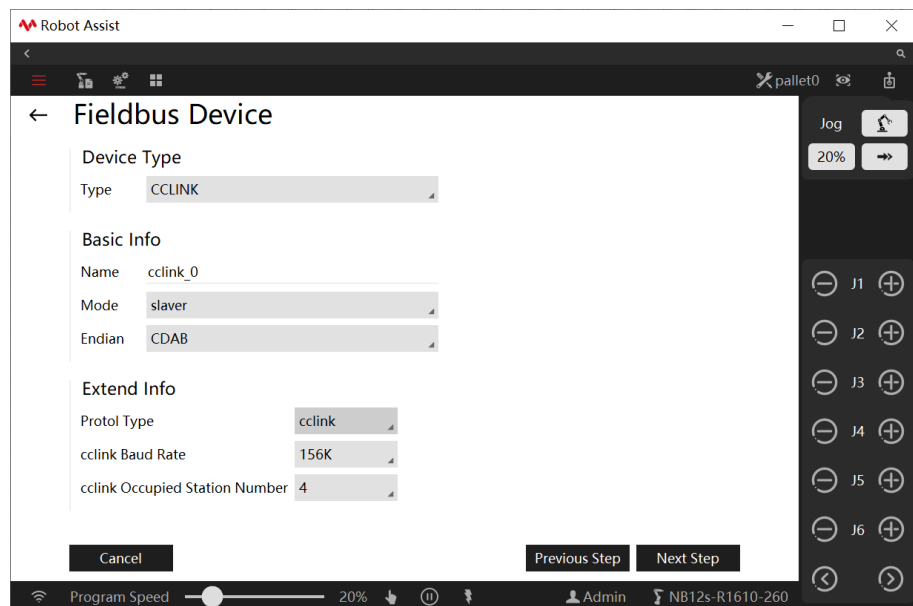
Parameter	Introduction
Mode	For CC-Link and CC-Link IE Field Basic protocol, the device can be configured as a slave only.
Protocol type	cclink refers to the EtherCAT expansion CC-Link module
cclink baud rate	Communication baud rate. Please note that the configuration of the master should match that of the slave
Number of cclink occupied stations	1 to 16 occupied stations can be configured. The default number is 4. Mode settings are recommended

7.3.3.2.2 CC-Link IE Field Basic configuration
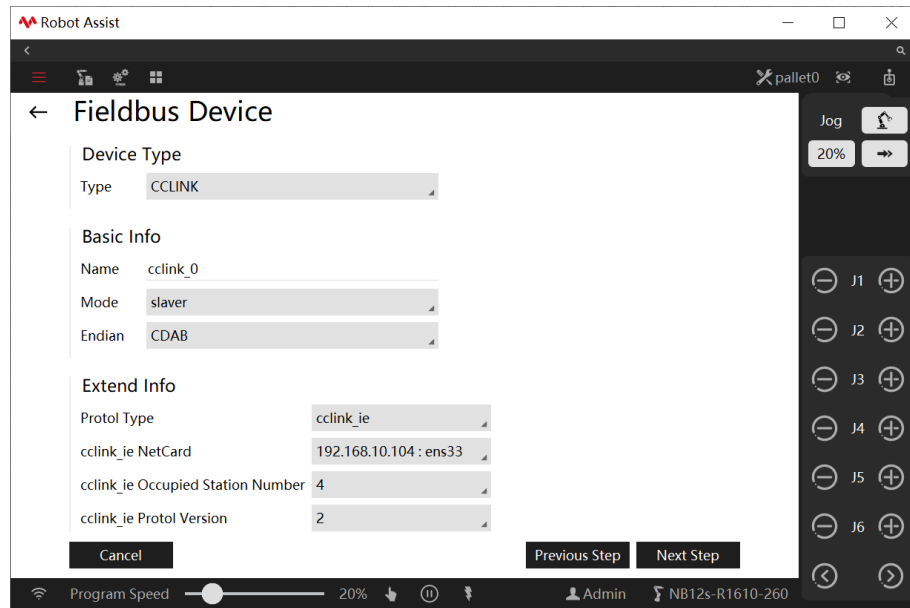
Parameter	Introduction
Protocol type	cclink_ie means the CC-Link IE Field Basic communication protocol that directly uses the robot's Ethernet port.
cclink_ie network card	Configure which Ethernet card is used for communication.
Number of cclink_ie occupied stations	1 to 16 occupied stations can be configured. The default number is 4. Mode settings are recommended
cclink_ie protocol version	Ver1 or Ver2 is optional. Please ensure that it is consistent with that of the master

7.3.3.3 EtherCAT communication

Click at the lower right corner on the bus device page to enter the interface to add a new communication bus device, and select the device type ETHERCAT. EtherCAT expansion devices can be used to access PROFINET and EtherNet/IP modules.
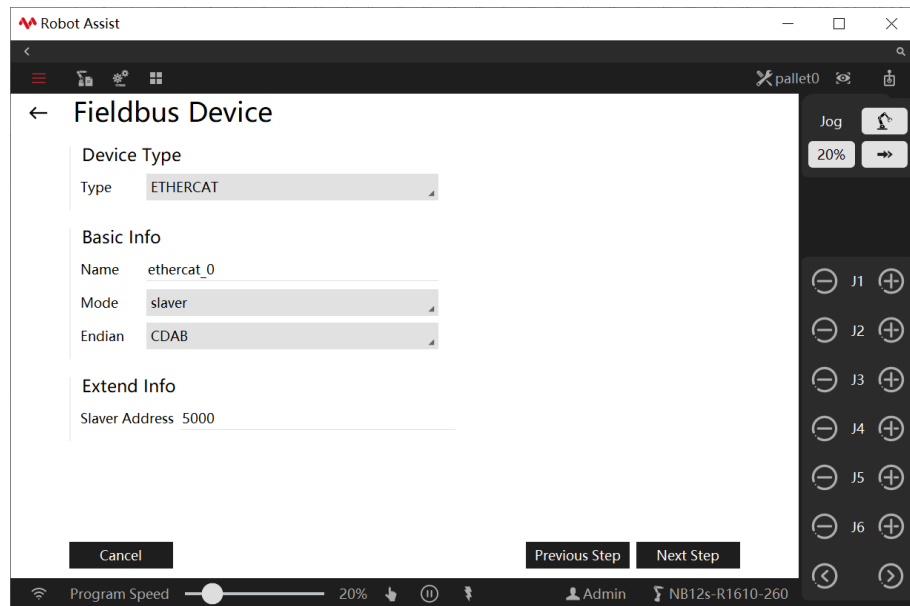
Slave address: The slave address number in the EtherCAT bus topology. Since the EtherCAT slave address number 1000-4000 is occupied by the robot internal devices, to avoid device address conflict, the EtherCAT slave address number of extended devices should not be less than 5000.

7.3.3.4 PROFINET communication

Configuration

Click at the lower right corner on the bus device page to enter the interface to add a new communication bus device, and select the device type PROFINET. The device can be configured as a slave only. One PROFINET slave can be configured for one robot, and multiple robots can join the same PROFINET network by modifying the PROFINET slave name to enable multiple slaves. The model selected for Slots 1-6 should be consistent with the correspondent-side configuration.
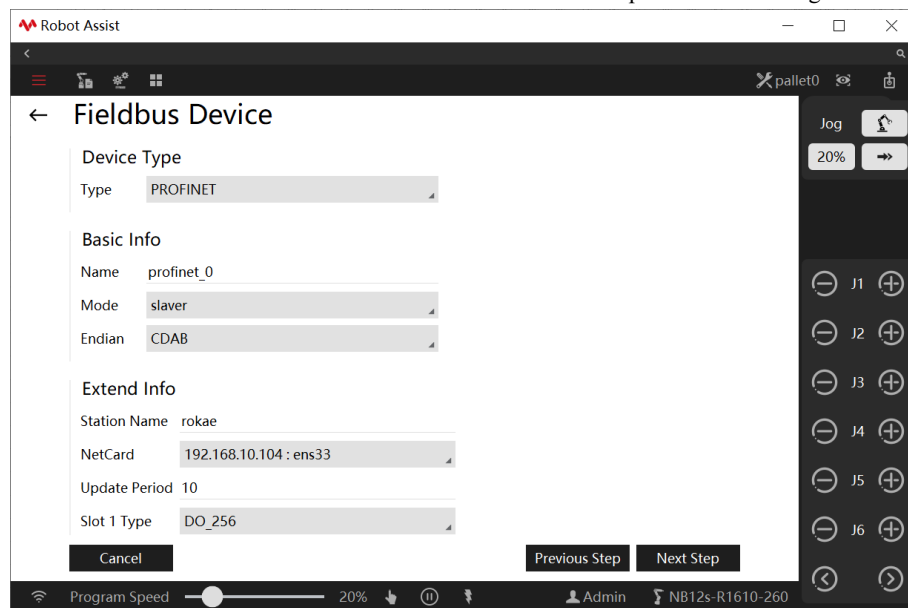
Parameter explanation:

Parameter	Description
Device type	PROFINET
Name	PROFINET slave name. It should be consistent withs the correspondent-side configuration.

	Chinese characters are not allowed
Mode	Only slaver is supported
Endianness	Select DCBA generally, depending on the agreement between the communicating parties
Network card	Select the network port to connect to the correspondent; includes the network card IP and name
Update period (ms)	Default to 10ms, minimum 2ms
Slot 1 type	Only DO_256 model can be selected, indicating that 256 digital quantities are output from the robot to the correspondent via slot 1
Slot 2 type	Only DI_256 model can be selected, indicating that there are 256 digital inputs from the correspondent to the robot via slot 2
Slot 3 type	The option models include AO_Int16_8/ AO_Int16_16/ AO_Int16_32/ AO_Int16_64/ AO_Int16_128/ AO_Int16_256. AO_Int16_8 means that there are 8 int16 analog outputs from the robot to the correspondent via slot 3, and so forth
Slot 4 type	The option models include AI_Int16_8/ AI_Int16_16/ AI_Int16_32/ AI_Int16_64/ AI_Int16_128/ AI_Int16_256. AI_Int16_8 means that there are 8 int16 analog inputs from the correspondent to the robot via slot 4, and so forth
Slot 5 type	The option models include AO_Float32_8/ AO_Float32_16/ AO_Float32_32/ AO_Float32_64/ AO_Float32_128/ AO_Float32_256. AO_Float32_8 means that there are 8 float32 analog outputs from the robot to the correspondent via slot 5, and so forth
Slot 6 type	The option models include AI_Float32_8/ AI_Float32_16/ AI_Float32_32/ AI_Float32_64/ AI_Float32_128/ AI_Float32_256. AI_Float32_8 means that there are 8 flaot32 analog inputs from the correspondent to the robot via slot 6, and so forth

7.3.4 Register

Explanation

The register is a type of variable of robot that can be used to exchange data with external devices so as to control the robot. The register can also be used as a variable in the current RL project. The register variables can be operated by commands or assignments. The register is a concept related to the robot itself, not a bus device. However, a specific register can be bound to a certain bus device for communication and data exchange. Specify the binding relationship when adding or editing a register. Each register occupies 2 bytes. For different types of variables, the number of registers occupied is different.

Configuration

Click Robot Configuration -> Communication -> Register and add register variables using the three buttons at the lower right corner of the interface.
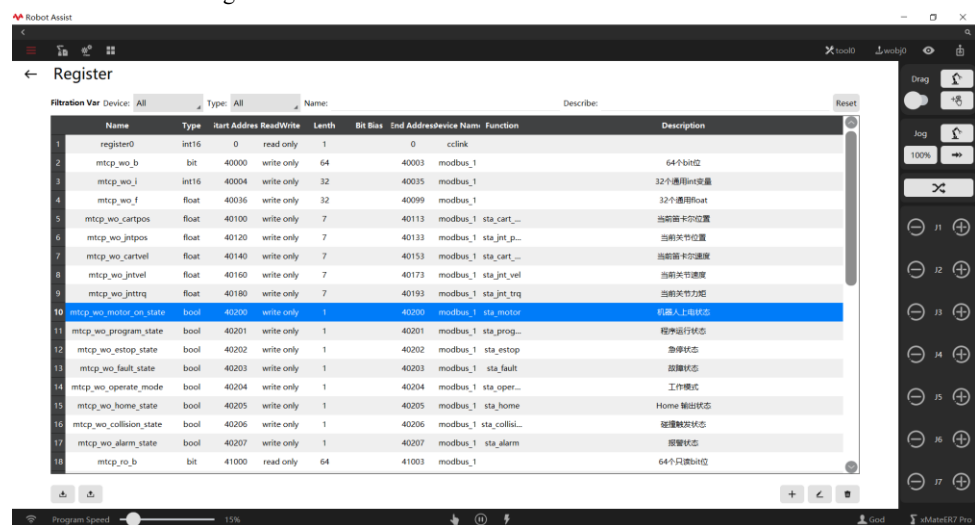
Name

The first column of the register list is the name of the registers, which is used for RL to access the register variables. Please note that the name on the list should be unique and should not be identical with that of the variables in any RL list, otherwise, there will be a variable conflict in the RL.

Type

The second column of the register list is the type of the registers, with four types optional: bit, bool, int16, and float. The details are as follows:

No.	Type	Description
1	bit	The bit type register variable occupies only one bit of a register, and the bit array needs to be an integer multiple of 16 bits. As shown in the figure above, for a bit type register starting from 41000-bit, a variable with a size of 64 occupies 4 registers from 41000 to 41003.
	bool	A bool variable occupies 1 register.
	int16	An int16 variable occupies 1 register.
	float	A float variable occupies 2 registers.

About bit type registers:
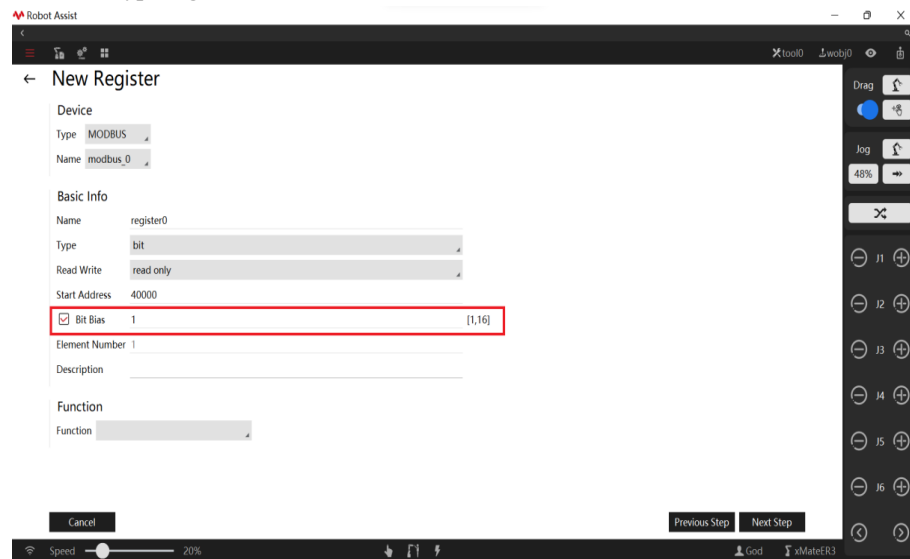
➢ As shown in the figure above, if the bit offset is checked, it means that a certain bit of a register is occupied, and the optional value is 1-16.

➢ For bit type registers, the function binding is only available when the bit offset is checked.

➢ For bit type registers, the number of elements is automatically modified to 1 when the bit offset is checked.

Parameter explanation

Parameter	Description
Initial register	The third column of the register list is the initial register, used to indicate the start address of the register. The address of all registers cannot be occupied repeatedly, otherwise, a register conflict error will occur. For example, if one register occupies 41000-41003, another register cannot start from 41002.
Read-write	The fourth column of the register list is the read-write attribute, indicating whether the register is read or written from the robot's perspective (not from the master or slave's perspective). For the state that the robot needs to output, it is a write-only register (when the robot serves as a slave, the holding register function code is 0x03; when the robot serves as a master, it is 0x06 or 0x10). For the command that the robot needs to receive from external devices, it is a read-only register (when the robot serves as a slave, the holding register function code is 0x06 or 0x10; when the robot serves as a master, it is 0x03).
Size	The fifth column of the register list is the size, indicating the number of the variables. For variables greater than 1, the variable can be referenced in the form of an array with the subscript starting from 1. Please note that it is different from the number of registers. As shown in the figure above, for registers 40140-40153, the variable type is float, and each float variable occupies 2 registers. Therefore, the size is 7, and the number of registers occupied is 2*7=14. The contents of the register can be referenced in the RL program via mtcp_wo_cartvel[1]~ mtcp_wo_cartvel[7].
Bit offset	The bit type register is mapped to the position of the register. Each register occupies two bytes, i.e. 16 bits. The bit offset refers to the position of the corresponding register, and the offset value is 1-16. If the bit offset is not checked when the bit type register is created, it is not displayed.
End register	The sixth column of the register list is the end register, used to indicate the last register address occupied by the register variable. When the register variables are arranged consecutively, this column will help users quickly plan the register assignment. For example, the start address of the next register can be determined by adding 1 to the value of this item.

Device name	The seventh column of the register list is the device name. The device name is defined when the bus device is created, indicating which bus device is bound to the register. The register can be bound to the CC-Link, CC-Link IE Field Basic, Modbus, and EtherCAT devices.	
Function	The eighth column of the register list is the function of the registers, with fixed function codes indicating the robot function of the register. The function codes are fixed and unmodifiable. The description is shown in the table below. There are two types of function codes: read-only and write-only. **Read-Only Function Codes** *(see table below)* **All system inputs of the above system registers are pulse-triggered. To ensure that the xCore system receives external commands correctly, please ensure that the pulse width of the external input is not less than 60 milliseconds.** **Write-Only Function Codes** *(see table below)*	

Read-Only Function Codes

ID	Function Code Name	Supported Binding Types	Function
1	Blank	N/A	No function, custom input
2	ctrl_clear_alarm	bit, int16, bool	Clear servo alarms
3	ctrl_estop_reset	bit, int16, bool	Emergency stop reset
4	ctrl_motor_on_off	bit, int16, bool	Motor power-on/power-off 1: Powered-on; 0: powered-off
5	ctrl_pptomain	bit, int16, bool	Program pointer to main
6	ctrl_program_start_stop	bit, int16, bool	Program running/stop
7	ctrl_set_program_speed	bit, int16, bool	Set program running rate
8	ctrl_switch_operation_auto_manual	bit, int16, bool	Switch between Automatic mode and Manual mode 1: Automatic mode; 0: Manual mode
9	ctrl_motoron_pptomain_start	bit, int16, bool	Power on, Pointer to main, and start program in order
10	ctrl_motoron_start	bit, int16, bool	Power on and start program in order
11	ext_cmd_set	bit, int16, bool	Remote control function: issuing commands See Remote Control
12	ext_reset	bit, int16, bool	Remote control function: overall function reset See Remote Control
13	ext_resp_get	bit, int16, bool	Remote control function: Acknowledge and clear the previous command response.
14	ext_request_data	int16 array	Remote control function: command function code. Array, register with a fixed size of 8.

All system inputs of the above system registers are pulse-triggered. To ensure that the xCore system receives external commands correctly, please ensure that the pulse width of the external input is not less than 60 milliseconds.

Write-Only Function Codes

ID	Function Code Name	Supported Binding Types	Function
1	Blank		No function, custom output
2	ext_error_code	int16	Remote control function: error code
3	ext_resp_set	bit, int16, bool	Remote control function: response after command execution
4	ext_response_data	int16 array	Remote control function: data to be fed back. Array, register with a fixed size of 8.
5	sta_alarm	bit, int16, bool	Servo alarm status 1: Servo alarm; 0: No alarm
6	sta_error_code	int16	Robot error code. Read error code = robot actual error code - 30000
7	sta_collision	bit, int16, bool	Collision detection state 1: Collision detected; 0: No collision

		8	sta_error_code	int16	Reported robot error code The error code is only a number. The actual error can be checked with the code (= reported robot error code - 30000).	
		9	sta_estop	bit, int16, bool	Emergency stop 1: Emergency stop currently triggered; 0: Normal	
		10	sta_home	bit, int16, bool	Whether the robot flange center is at home 1: At home; 0: Not at home	
		11	sta_motor	bit, int16, bool	Motor power status 1: Powered-on; 0: Powered-off	
		12	sta_operation_mode	bit, int16, bool	Current operating mode 1: Automatic mode; 0: Manual mode	
		13	sta_program	bit, int16, bool	Whether the program is currently running 1: Program running; 0: Idle	
		14	sta_program_speed	int16	Query the current program running speed (in percentage terms).	
		15	sta_cart_pose	float array	Query the current Cartesian space pose of the robot. Requirements for bound registers: float array, size - 8	
		16	sta_jnt_pose	float array	Query the current joint angle of the robot. Requirements for bound registers: float array, size - 8	
		17	sta_jnt_trq	float array	Query the current joint torque of the robot. Requirements for bound registers: float array, size - 8, unit: N.m;	
		18	sta_jnt_vel	float array	Query the current joint velocity of the robot. Requirements for bound registers: float array, size - 8, unit: rad/s;	
		19	sta_robot_is_busy	int,bit,bool	Whether the robot is executing time-consuming operations such as pptomain: 1: Executing; 0: Idle	
		20	sta_tcp_pose	float array	Pose of the robot TCP. Requirements for bound registers: float array, size - 7	
		21	sta_tcp_vel	float array	Speed of the robot TCP. Requirements for bound registers: float array, size - 7	
		22	sta_tcp_vel_mag	float	Robot TCP combined linear velocity	

How to Use

The control system reads and modifies the registers in two ways: command or assignment.

For command type, two commands - WriteRegByName and ReadRegByName - are provided. The assignment type is more intuitive and simple with the operator "=".

1) Command type:

➢ WriteRegByName(modbus_reg[index], rl_symbol)

The first parameter is the register name configured in Robot -> Register. [index] can be used to offset the start address of the corresponding register. Limitation: 1 <= index <= the maximum size of the register. The default index is 1.

The data in the control system (such as the number of cycles in the RL language) can be output to its bound devices through registers. Assume it is defined as "int rl_value" in the control system. If you want to output it to an external device, you can specify a register, such as the first register of "mtcp_wo_i" in the default configuration. Simply add a WriteRegByName command in the RL language, and the parameter will be sent to the external device via Register - Bus Device.
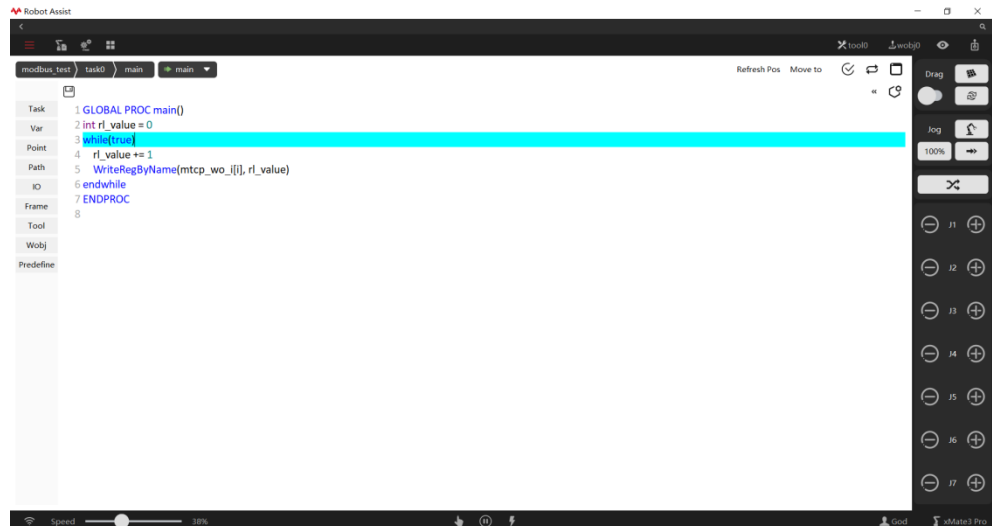
> ➤ ReadRegByName(modbus_reg[index], rl_symbol)

Similar to the WriteRegByName command, it can update the value in the register to the variable of the RL program, and therefore can be used to control the execution process of the RL program and kinematic parameters, etc.

2) Assignment type

Directly use the operator "=". For example, "mtcp_wo_i[1] = 1" is to update the value of the first element of the register mtcp_wo_i to 1. Similarly, "a = mtcp_wo_i[1]" is to update the value of the first element of the register mtcp_wo_i to the variable a of the RL program.

Register retain configuration

Register retain: Creates register a with the hold property whose current value is held on a non-volatile storage medium when the robot restarts, shuts down, powers off, or when RL is stopped. When the robot powers on again or RL is running again, the value of register a is restored to the value held before the robot shuts down or RL is stopped.

The register retain configuration interface is shown below:
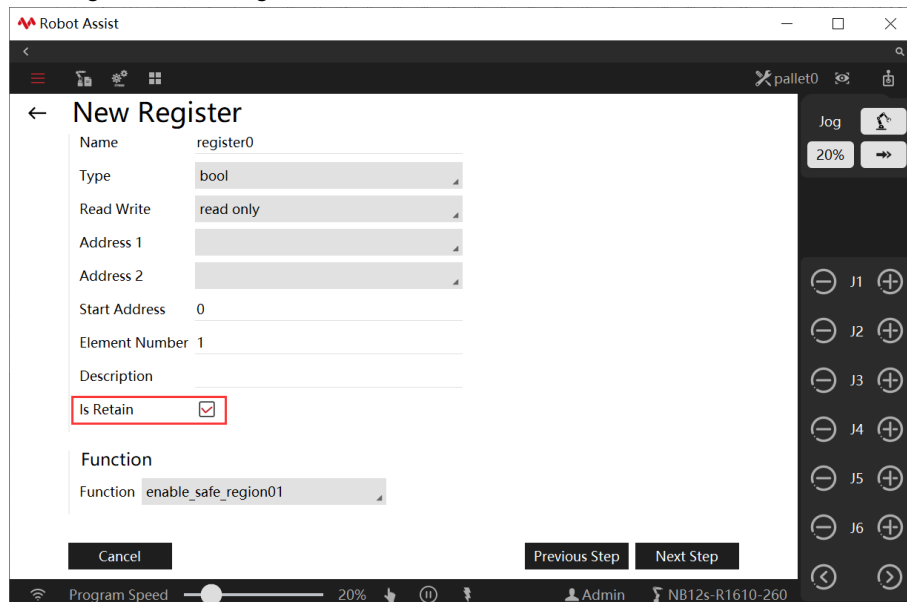
Conflict checking during register import

The register address cannot be the same for the same register attribute (read and write) on one device. If the address is the same, the newly imported one prevails, while the original conflicting register is overwritten. A pop-up window prompts the user to choose whether to replace the current register. When creating the 7 registers starting with ext: ext_cmd_set, ext_resp_set, ext_resp_get, ext_reset,

ext_response_data, ext_request_data, ext_error_code, if the register has been bound by the register address, these addresses cannot be bound by another register. When importing the above 7 registers, if the function codes have already been bound in the HMI, and the newly imported register list also involves such function codes, the newly imported ones will prevail, and the original conflicting register will be overwritten. A pop-up window will prompt the user to choose whether to replace the current register.

7.3.5 IO device

Explanation

IO includes DI, DO, AI, and AO. There are three types of signal sources: controller cabinet built-in, EtherCAT expansion, and field bus expansion. For industrial robots, the controller cabinet has several built-in DIs and DOs. For cobots, the base and the end-effector have several built-in DIs and DOs. For industrial robots, the EtherCAT expansion interfaces are reserved on the controller cabinet to connect EtherCAT expansion modules to generate new DI, DO, AI, and AO. The Modbus bus expansion can also be configured with IOs.
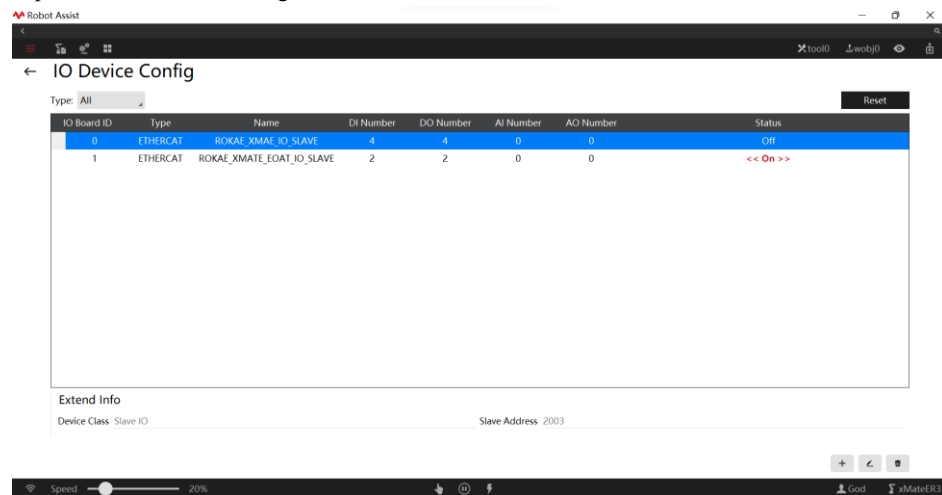
Parameter

Click at the lower right corner on the IO device configuration page to enter the IO device configuration interface. The parameters on the interface may vary with the device type.
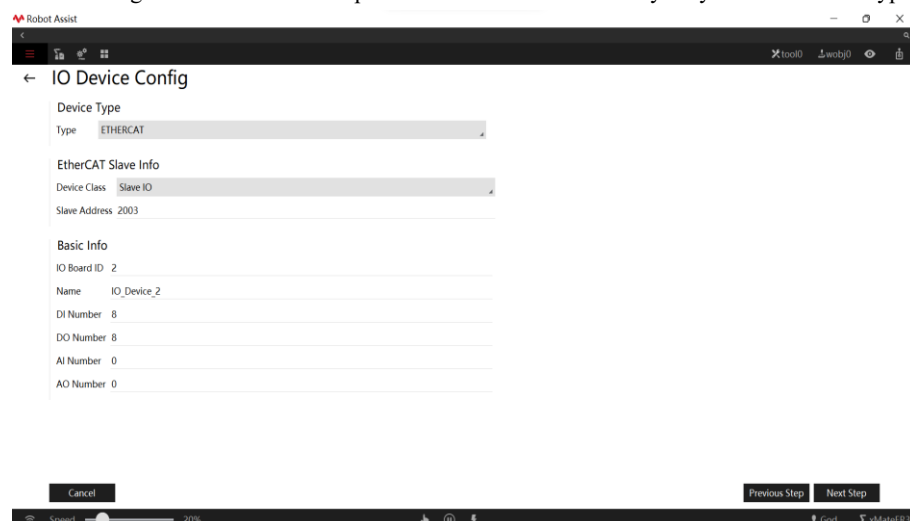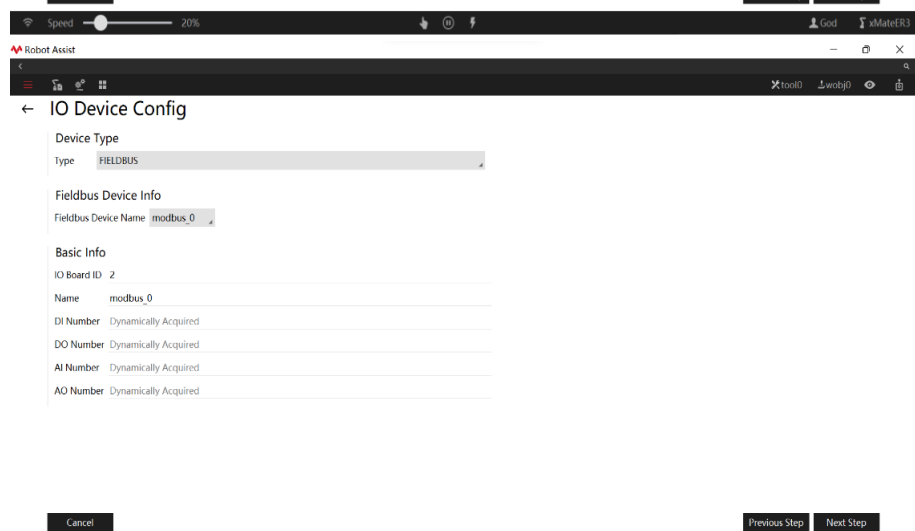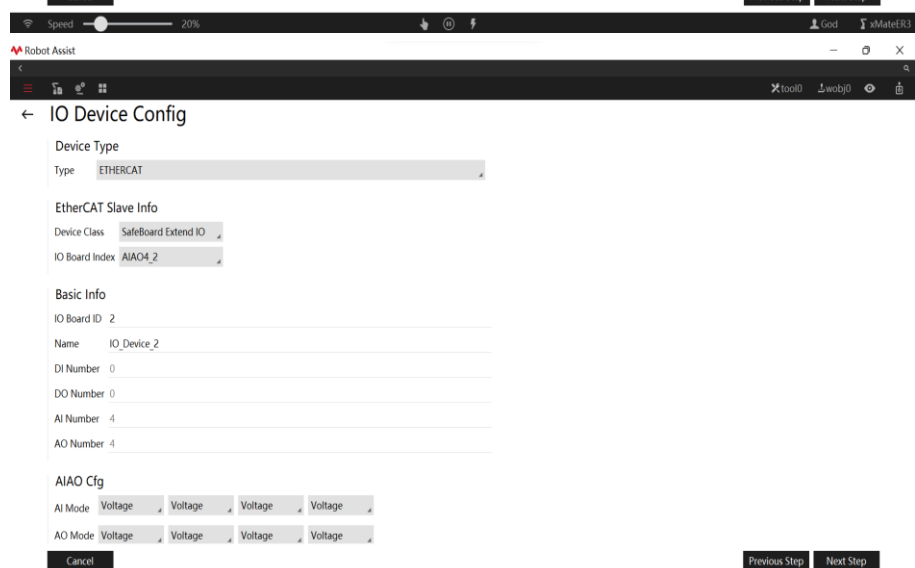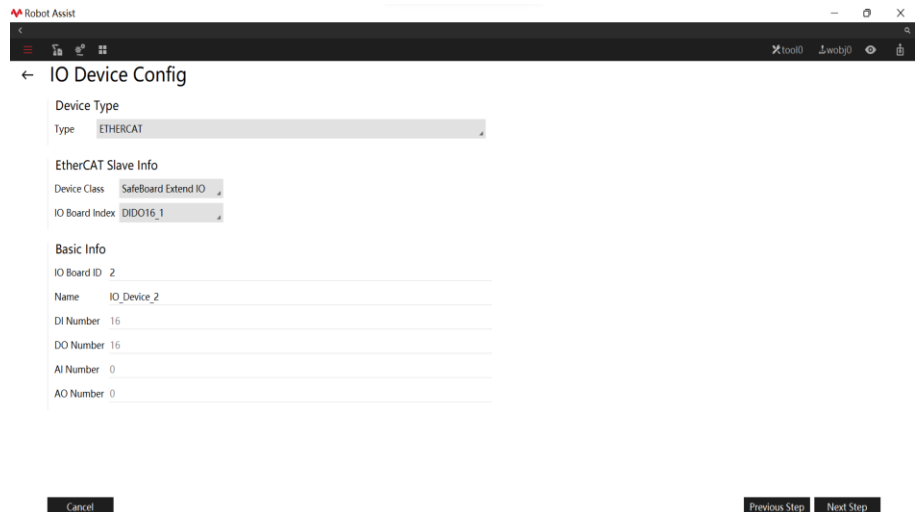
Parameter	Description
Device type	EtherCAT and FIELDBUS are optional. EtherCAT refers to IO expansion with the EtherCAT bus and expansion modules. The expansion modules can only serve as slaves, and the slave address needs to be configured.
EtherCAT slave information - Device type	SafeBoard IO, SafeBoard Extend IO, xPanel IO, and Slave IO are optional. SafeBoard IO refers to the DI and DO on the robot safeboard. SafeBoard Extend IO refers to the expansion IO on the robot safeboard, generally the expansion IO on the safeboard in the XBC_5 controller

	cabinet. xPanel IO refers to the DI and DO of the end-effectors of the cobots. Slave IO refers to the EtherCAT expansion module.
EtherCAT slave information - IO board type	When SafeBoard Extend IO is selected in EtherCAT slave information - Device type, the option IO board type appears for selection of the safeboard expansion IO board. DIO16_1, DIO16_4, DIO16_5, DIO16_6, AIAO4_2, and AIAO4_3 are optional. The last digit of the option refers to the address of the safeboard expansion IO board. Select the safeboard expansion IO board that is actually connected. In addition to manual editing by the user, the controller can automatically identify and add the safeboard expansion IO board.
EtherCAT slave information - Slave address	The slave address of the expansion module in the EtherCAT bus topology. It should not conflict with the address of the safeboards, joints, or the cobot end-effectors.
FIELDBUS - Bus device name	The custom name when a bus connection is created on the Bus Device page. It is used to associate with the Bus Device.
IO board serial number	A virtual IO board is generated for each IO device configuration for the control system to classify and manage the IO boards internally. The IO board serial number is the unique number for virtual IO board management.
Name	The custom name of the virtual IO board. It is used for filtering in Status Monitoring -> IO Signal.
Number of digital inputs	Number of DIs.
Number of digital outputs	Number of DOs.
Number of analog inputs	Number of AIs.
Number of analog outputs	Number of AOs.
Analog Quantity IO Configuration	When SafeBoard Extend IO is selected in EtherCAT slave information - Device type and AIAO4_2 or AIAO4_3 is selected in IO board type, the option Analog IO Configuration appears. Each analog channel can be configured as voltage type or current type.

Status Monitoring

Monitor the created DI, DO, AI, and AO in Status Monitoring -> IO Signal. The IO signals can be filtered by Virtual IO Board Name. Only the IO signals currently configured on the virtual IO board will be displayed. You can also filter the signals by signal type. Only a certain type of DI, DO, AI, and AO signals will be displayed.
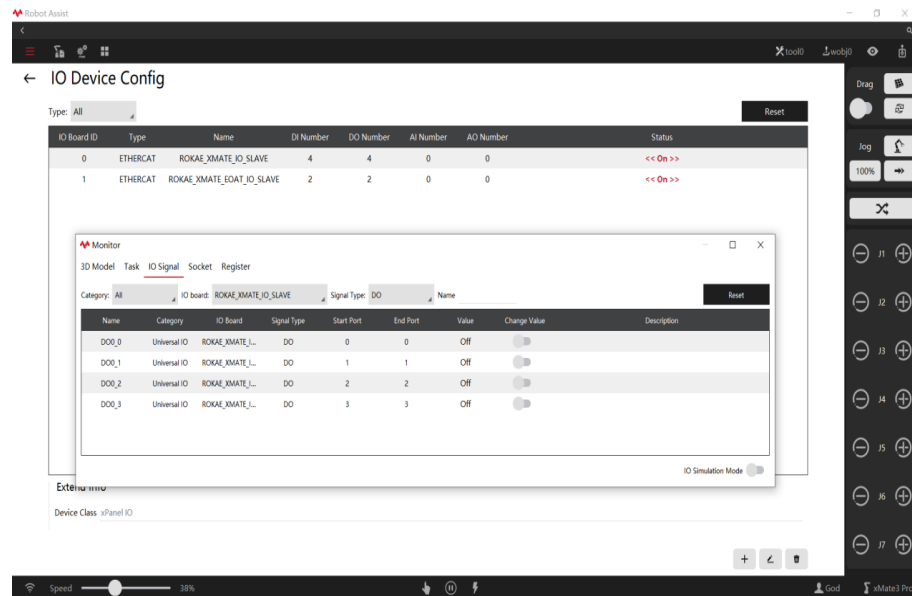
How to Use

After the virtual IO board is configured, a default name will be generated for the IO signal. The default name can be used directly in the RL program. Or you can bind and rename the created IO signal in Project -> IO Signal, and use the new name in RL.

Use default name

The board IO_Device_6 generates the DI6_X IO signals by default. As shown in the figure, DI6_0 is processed directly in the RL program.
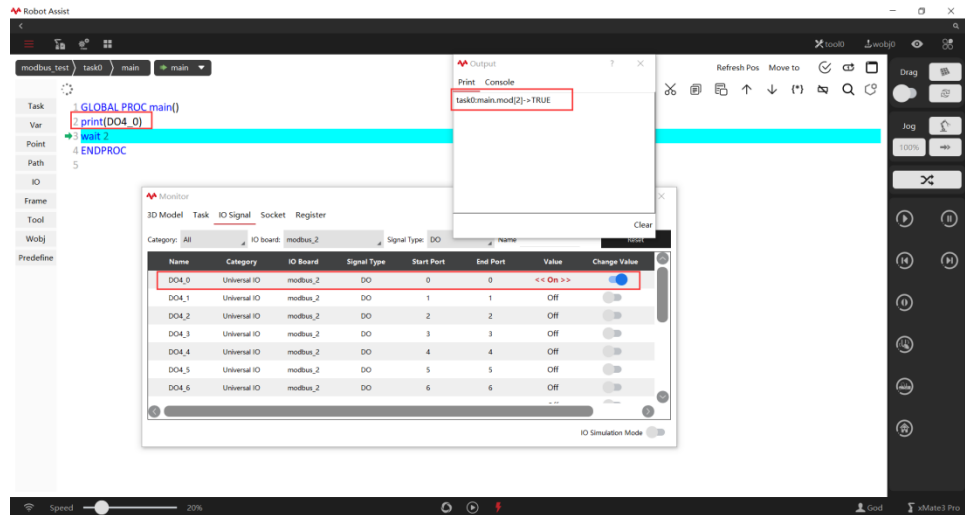
Bind and rename in RL project
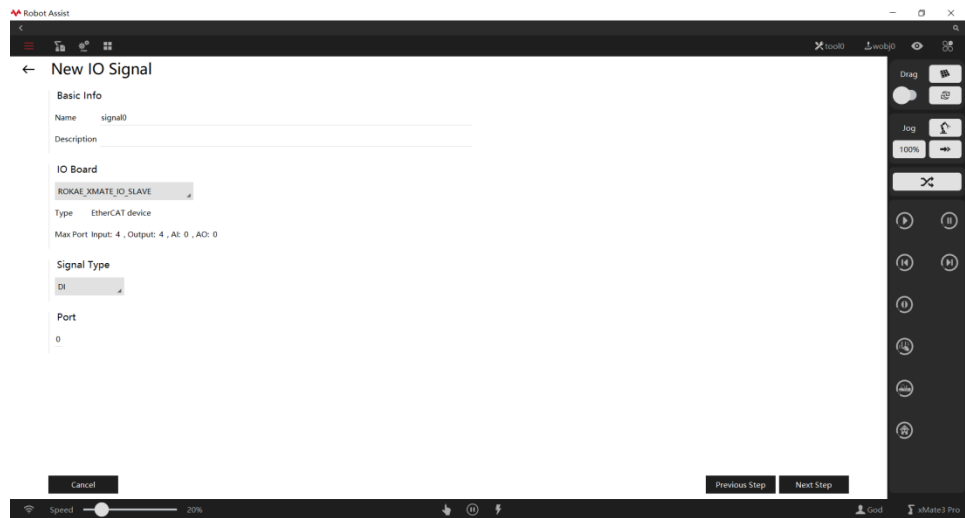
7.3.5.1 Register remote control

Explanation

Remote control is a combination function performed with registers of 7 different functions. It is used to achieve complex business logic interactions in a specific sequence. External devices can fulfill functions such as robot Jog, updating point position, obtaining robot position and status, etc. via the remote control function.

Register function

External devices use four types of registers to control the robot. These registers are read-only for the robot.

No.	Function Code Name	Attribute	Type	Size	Function
1	ext_cmd_set	Read-only	int16, bool, bit	1	Issuing commands 1. Set ext_cmd_set to 1 to send a request for command execution. The request is responded only when ext_cmd_set is set to 1. 2. To avoid misoperation, be sure to set the command data to the data area before execution. (The command data is temporarily stored in the cache and is responded only when ext_cmd_set is 1). 3. After the command is executed, clear ext_cmd_set (set it to 0).

2	ext_reset	Read-only	int16, bool, bit	1	Function reset: 1. The signal is used to enable the remote control function. Always keep the register state at 1 when using the function. 2. The function stops when the register state is 0. 3. The signal is also used for commands to reset or interrupt the action when the interface function is abnormal.
3	ext_resp_get	Read-only	int16, bool, bit	1	Acknowledge and clear the previous command response, and reset ext_resp_set to 0.
4	ext_request_data	Read-only	int16	8	Command function code. Array, register with a fixed size of 8. For details, refer to the introduction in the function code section.

External devices use three types of registers to obtain the robot status. These registers are write-only for the robot.

No.	Function Code Name	Attribute	Type	Size	Function
1	ext_error_code	Write-only	int16	1	Remote control function: error code
2	ext_resp_set	Write-only	int16, bool, bit	1	After responding to the control command, the robot sets the register to 1, indicating that the command is executed.
3	ext_response_data	Write-only	int16	8	Remote control function: data to be fed back. Array, register with a fixed size of 8.

Procedure

The combined use of 7 types of registers and control flow are shown in the figure below:

```
        ┌──────────┐
        │  Start   │
        └────┬─────┘
             │
   ┌─────────────────────┐
   │ Initialize register:│
   │    ext_reset = 0    │
   │   ext_cmd_set = 0   │
   └──────────┬──────────┘
              │
   ┌─────────────────────┐
   │ Enable remote control:│
   │   Keep ext_reset = 1  │
   └──────────┬──────────┘
              │
   ┌─────────────────────┐
   │   Prepare command:  │
   │        Set          │
   │   ext_request_data  │
   │       [0]-[7]       │
   └──────────┬──────────┘
              │
   ┌─────────────────────┐
   │   Issue command:    │
   │  Keep ext_cmd_set=1 │
   └──────────┬──────────┘
```

Normal Control Process

```
        ┌──────────────┐
        │Error occurred│
        └──────┬───────┘
               │
   ┌─────────────────────┐
   │   Reset command:    │
   │    ext_reset = 0    │
   └──────────┬──────────┘
              │
   ┌─────────────────────┐
   │ Disable command issuing:│
   │    ext_cmd_set = 0  │
   └──────────┬──────────┘
              │
   ┌─────────────────────┐
   │ Enable remote control again:│
   │    ext_reset = 1    │
   └──────────┬──────────┘
              │
        ┌──────────────┐
        │Normal Control│
        │   Process    │
        └──────────────┘
```

Error Handling Process

Wait for instruction processing result: Query ext_resp_set = 1?

Read response: Read ext_response_data [0]~[7]

Confirm that the result is read: Keep ext_resp_get = 1 the communication cycle of at least one register

Confirm that the result is read: ext_resp_get = 0 ext_cmd_set = 0

Keep controlling?

End

Command format

Commands and responses are implemented with 8 registers individually.

The command signal ext_request_data (eight registers occupied: reg0 - reg7) is used to specify the data area of the commands and relevant parameters. A command consists of multiple characters:
1) Character: a 16-bit register.
2) Command format: a command consists of up to 8 characters and varies with the command. The shortest command consists of 1 character.

Command No.	Command No. 1	Command No. 2	……	Command No. 7

The response signal ext_response_data (eight registers occupied: reg0 - reg7) is used to obtain the data area of the responses. A response consists of multiple characters:

1) Character: a 16-bit register.

2) Response format: a response consists of up to 8 characters and varies with the received command. The shortest response consists of 1 character. However, an abnormal response always occupies 3 characters.

Command No.	Response No. 1	Response No. 2	……	Response No. 7

The available command numbers are shown in the table below:

Command Type	Description	Command Code	Command Length	
			Command	Response
JOG	Set Jog space	1	2	3
	Obtain Jog space	2	1	4
	Set Jog speed	3	2	3
	Obtain Jog speed	4	1	4
	Set Jog step length	5	2	3
	Obtain Jog step length	6	1	4
	Start Jog	7	4	2
	Stop Jog (without parameters)	8	1	2
	Update point position	9	2	2
	Move to point position	10	2	2
Setting information	Set tools	11	2	3
	Obtain current tool id	12	1	4
	Set work object	13	2	3
	Obtain current work object id	14	1	4

Command description

1) Set Jog space:

Command/Reply	Command Code	Parameter 1	Parameter 2
Set Jog space	1	Frame: 1: Joint space 2: World frame 3: Flange frame 4: Base frame 5: Tool frame 6: Work object frame	N/A
Reply	1	Result: 0 - Succeed; 1 - Fail.	Error Code

2) Obtain Jog space:

Command	Command Code	Parameter 1	Parameter 2	Parameter 3
Obtain Jog space	2	N/A	N/A	N/A
Reply	2	Result: 0 - Succeed; 1 - Fail	Error Code	Frame: 1: Joint space 2: World frame 3: Flange frame 4: Base frame 5: Tool frame 6: Work object frame

3) Set Jog speed:

Command/Reply	Command Code	Parameter 1	Parameter 2
Set Jog speed	3	Jog speed (1-100)	N/A
Reply	3	Result: 0 - Succeed; 1 - Fail.	Error Code

4) Obtain Jog speed:

Command	Command Code	Parameter 1	Parameter 2	Parameter 3
Obtain Jog speed	4	N/A	N/A	N/A
Reply	4	Result: 0 - Succeed; 1 -	Error Code	Jog speed (1-100)

		Fail		

5) Set Jog step length:

Command/Reply	Command Code	Parameter 1	Parameter 2
Set Jog step length	5	1: Continuous 2: 10 mm step length 3: 1 mm step length 4: 0.1 mm step length 5: 0.01 mm step length	N/A
Reply	5	Result: 0 - Succeed; 1 - Fail.	Error Code

6) Obtain Jog step length:

Command	Command Code	Parameter 1	Parameter 2	Parameter 3
Obtain Jog step length	6	N/A	N/A	N/A
Reply	6	Result: 0 - Succeed; 1 - Fail	Error Code	1: Continuous 2: 10 mm step length 3: 1 mm step length 4: 0.1 mm step length 5: 0.01 mm step length

7) Start Jog:

The command is dependent on command code 1: set Jog space. In joint space, the value of parameter 1 represents the joint number (J1-J7: 1 for J1, ..., 7 for J7); in Cartesian space, it represents the (x, y, z, a, b, c, elb) number (1 for x, ..., 7 for elb).

Command/Reply	Command Code	Parameter 1	Parameter 2
Start Jog	7	Operation mode: Joint space - representing joint number; Cartesian space - representing (x, y, z, a, b, c, elb)	Jog direction: 1: negative 2: positive
Reply	7	Result: 0 - Succeed; 1 - Fail.	Error Code

8) Stop Jog:

Command/Reply	Command Code	Parameter 1
Stop Jog	8	N/A
Reply	8	Result: 0 - Succeed; 1 - Fail.

9) Update point position:

Command/Reply	Command Code	Parameter 1	Parameter 2
Update point position	9	Number in the RL project point list	N/A
Reply	9	Result: 0 - Succeed; 1 - Fail.	Error Code

10) Move to point position:

Command/Reply	Command Code	Parameter 1	Parameter 2
Move to point position	10	Motion mode: 1: MoveAbsj; 2: MoveJ; 3: MoveL	Number in the RL project point list
Reply	10	Result: 0 - Succeed; 1 - Fail.	Error Code

11) Set current tool:

Command/Reply	Command Code	Parameter 1	Parameter 2
Set current tools	11	Number in the RL project tool list	N/A
Reply	11	Result: 0 - Succeed; 1 - Fail.	Error Code

12) Obtain current tool id:

Command/Reply	Command Code	Parameter 1	Parameter 2	Parameter 3
Obtain current tool id	12	N/A	N/A	N/A

| Reply | 12 | Result: 0 - Succeed; 1 - Fail. | Error Code | Current tool id |

13) Set current work object:

Command/Reply	Command Code	Parameter 1	Parameter 2
Set current work object	13	Number in the RL project work object list	N/A
Reply	13	Result: 0 - Succeed; 1 - Fail.	Error Code

14) Obtain current work object id:

Command/Reply	Command Code	Parameter 1	Parameter 2	Parameter 3
Obtain current work object id	14	N/A	N/A	N/A
Reply	14	Result: 0 - Succeed; 1 - Fail.	Error Code	Current work object id

Error Code

During command configuration, parameter errors, robot status mismatch, or other conditions may lead to configuration failure. Error codes can be used to check the robot's problems in this case.

The control system has three types of error codes:

1) ext_response_data: error code of command execution results.

2) ext_error_code: The command cannot be executed, for example, the robot is busy or the remote control flag bit is incorrect, etc.

3) sta_error_code: the robot error code. Read the register when an error occurs during Jog.

Normally, the error code should be used according to the following steps:

1) After sending the execution command (ext_cmd_set=1), first read ext_error_code. If there is no error code, read the return value of ext_response_data. If the return value is not zero, read the error code of ext_response_data.

2) For motion operations (Jog and move to point position), if the above return values are both 0, read sta_error_code to see if there is a stop in the motion caused by an error (such as singularity and overrun).

ext_error_code description:

Error Code	Meaning	Remarks
01	Unsupported command	
02	Invalid parameter	
03	Incorrect control flag bit	Check whether ext_resp_set is 0 or 1.
04	Robot busy	The robot is executing a command and is forbidden to respond to others.
05	No corresponding number found	Tool, point position, and work object id
06	Unmatched point type and motion type	The point type does not match the motion type for the "Move to point position" command. For example, only the MoveAbsJ command can be used for joint space points, and only the MoveJ or MoveL command can be used for Cartesian space points.
07	The number of axes entered does not match the model	
11	Incorrect Manual/Auto Mode	
12	Incorrect robot status. Please check if the robot is in Jog Mode.	The robot can only be jogged in Jog Mode and can not be jogged in non-Jog Mode such as Drag Mode.
13	Incorrect power-on status	The robot can only be jogged when powered on.
14	The robot is in non-position mode and can not be jogged	Similar to error code 12.
15	Report algorithm error when unable to start Jog	The error is reported when the robot cannot be jogged for various reasons.
20	Encounter singularity	
21	Moved to target point	If the robot moves to a point it has reached earlier, an error occurs.

7.3.5.2 Modbus expansion IO

Explanation

When real IO electrical signals are required to interact with external devices, it is recommended to use an adapter module. Connect the adapter module to the controller cabinet, and operate the adapter module via the field bus supported by the control system. You are advised to choose Modbus IO modules recommended by ROKAE. The module is a Modbus TCP slave and controls the robot through the coil function. Therefore, the robot needs to be configured as a Modbus Master with the coil function enabled.

Parameter configuration

According to the configuration method of the field bus and expansion IOs, configure Bus Device first and then configure IO Device when using Modbus expansion module to expand real IOs.
1) Configure Bus Device
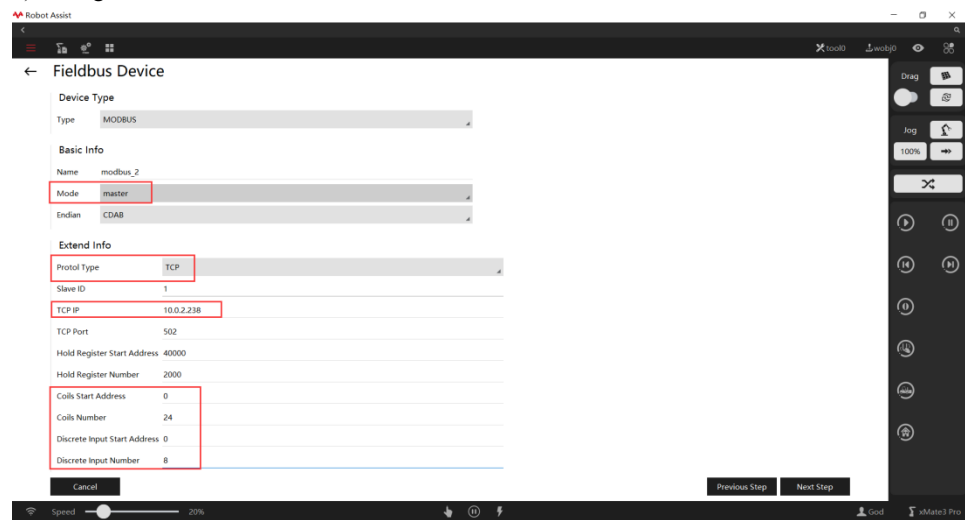
Parameter	Value/Description
Master-slave mode	Select "Master" to use the robot as a master.
IP	When the robot is the master, fill in the IP of the Modbus IO module.
Port number	When the robot is the master, fill in the port number of the Modbus IO module.
Slave ID	Modbus IO module Slave ID
Coil status	Modbus module "coil status" register. Range: 1 - 32, data type: int.
Input status	Modbus module "input status" register. Range: 1 - 32, data type: int.
Connect	After the parameters are configured, click "Connect" to complete the configuration.

2) Configure IO Device Configuration
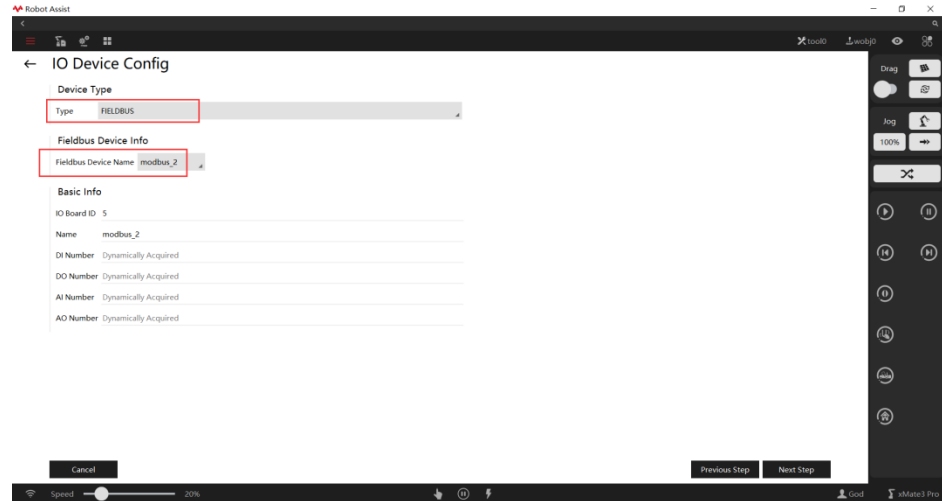
Parameter	Value/Description
Device type	Select FIELDBUS.
Bus device name	Select the device name defined in the Bus Device configuration.
Basic information	Digital I/O and analog I/O configured in Bus Device configuration will be obtained automatically. No configuration is required.

3) Enabling function and status monitoring

Please note that after the Bus Device and IO Device Configuration are just configured, the configured expansion IOs will not be displayed in the status monitoring because the bus connection is not enabled yet. To use these IOs, enable the bus connection to correctly establish connection and communication between the controller cabinet and the expansion IO modules.
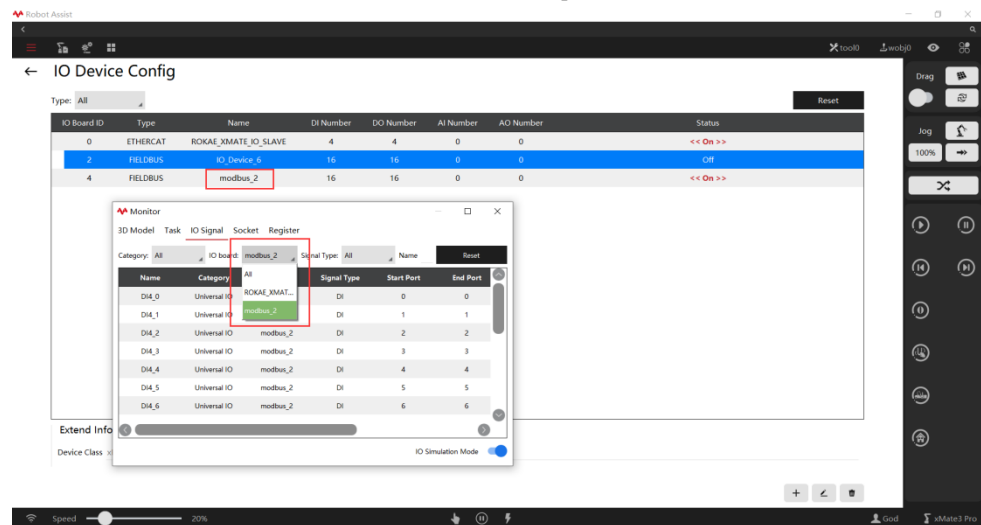
As shown in the figure, after the bus device is enabled, the connection between the controller and the expansion modules is established and works properly. Now the IO of the bus device modbus_2 is displayed in the status monitoring of the IO signals, and the number of DIs and the total number of DIs and DOs remains the same as the bus device configuration.
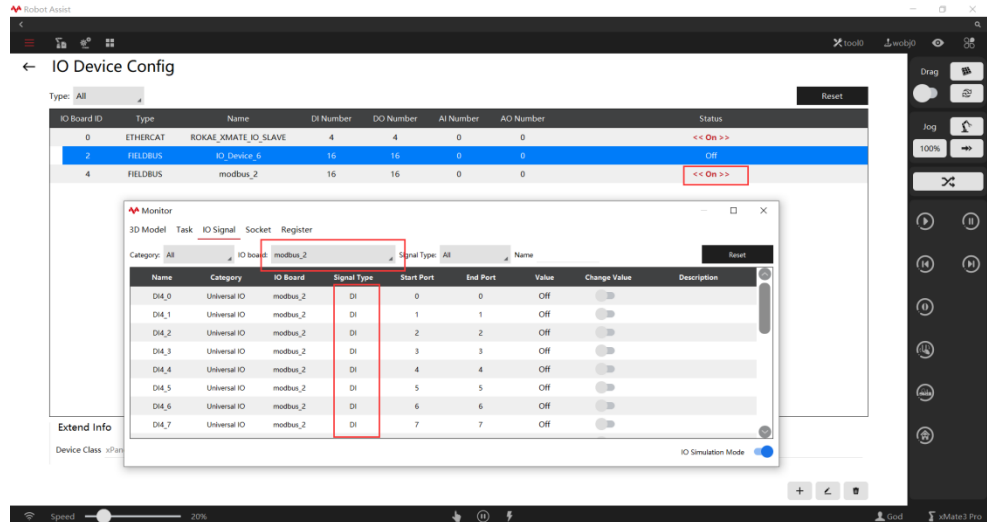
7.3.6 Serial Communication

Explanation

Users can use serial ports to communicate with external devices. The use of serial ports requires hardware support. For industrial robots, an RS-232 serial port is reserved on the XBC5 controller cabinet. Or the USB interface on the controller cabinet can be used for serial communication via a USB-to-RS-232 interface module. The cobots do not reserve relevant interfaces, and therefore do not support serial communication.

Configurations

Before using the serial port, configure the following parameters: serial port name (used in RL), serial port, baud rate, data bit, stop bit, and parity bit. Go to the configuration interface via Robot Configuration -> Communication -> Serial Port Configuration, as shown in the figure below. Please try to ensure that the parameter settings at both ends of the serial communication are consistent. Otherwise, errors may occur in sending and receiving data.
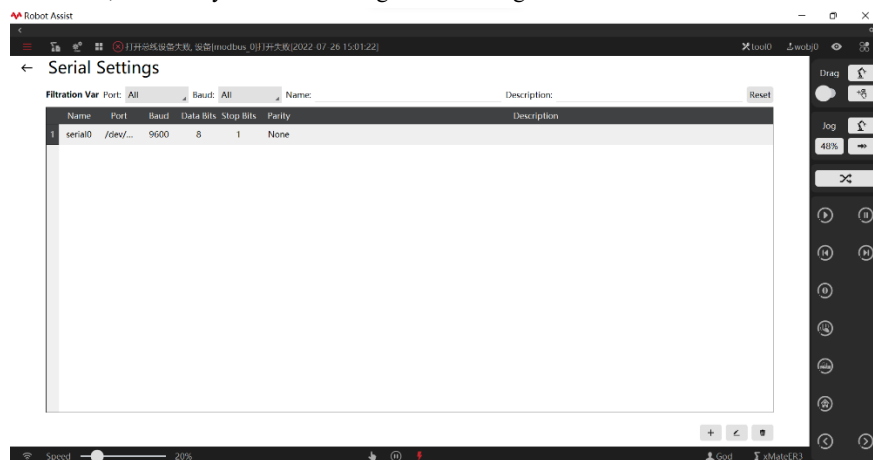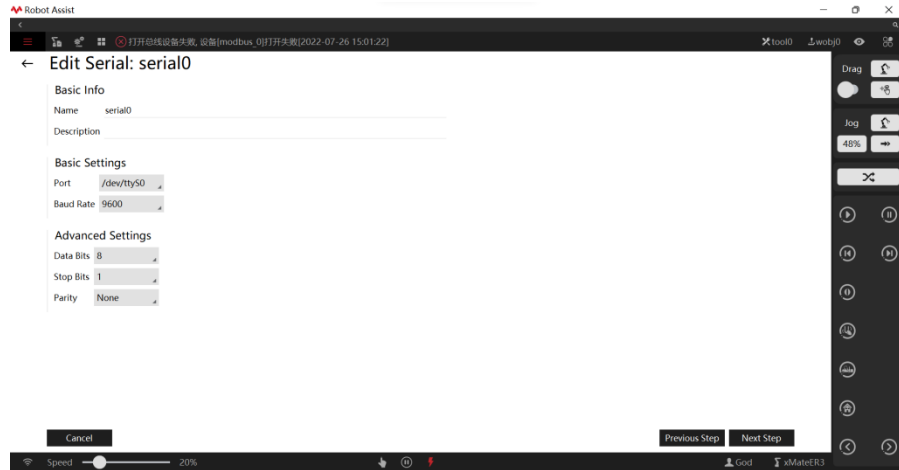
Parameter	Description
Name	The custom name to be used as the unique identifier in RL to use the serial port resources. Please note the serial port name is subject to the name conflict restriction in the project. It should not be identical with the existing network identifiers in the project or the existing identifiers of other serial ports.
Port	System port. The control system lists all the serial port resources detected (including the USB-to-RS-232 ports) for users' selection and use. This is the name of the serial port resources detected by the operating system.
Baud rate	1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 are optional.
Data bit	5, 6, 7, and 8 bits are optional.
Stop bit	1, 1.5, and 2 bits are optional.
Parity bit	Odd parity, Even parity, Mark parity, Space parity, and None parity are optional.

Use Serial Ports

After configuring the serial port, you can use the serial port interface in the RL program without restarting. The serial port function involves a series of commands. For details, please refer to the section about serial port commands in RL Commands.

7.3.7 End-effector tool communication

Explanation

The xMate robot can control the opening and closing of DH grippers, and the end-effector tool interface supports IO communication and RS485 communication.
The function is only applicable to cobots, including the xMate ER series and xMate CR series.

Parameter setting
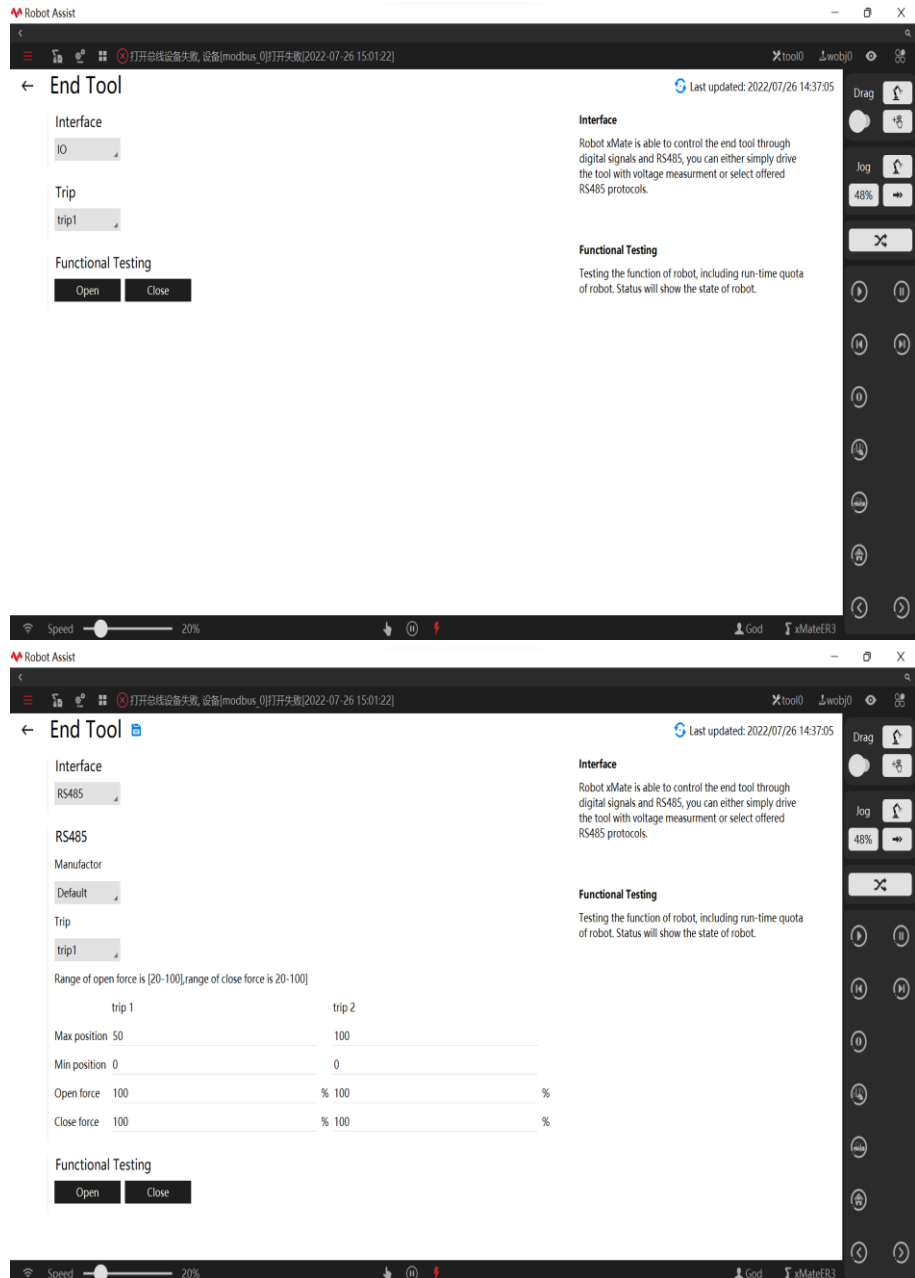
Parameter setting	Value/Description
Interface	Communication protocol, optional controllable via IO or RS485.
Path	It includes two sets of travel attributes trip1 and trip2, which contain the opening/closing position and force.
Maximum position	Maximum opening position, unit: percentage
Minimum position	Minimum opening position, unit: percentage
Supporting force	The force used when the gripper is opened, unit: percentage
Gripping force	The force used when the gripper is closed, unit: percentage

---

i Notes

RS485 supports setting of the gripper trip parameter. For IO control, the trip parameters can only be set through the DH communication adapter.

7.3.8 Electric gripper and suction cup

Explanation

xMate CR and ER robots equipped with end-effectors support RS485 communication and are adapted to Jodell electric grippers and suction cups. This interface is used to test the basic functions of the electric gripper and suction cup.

This function is only available for cobots of the xMate ER series and the xMate CR series. Before use, the user needs to confirm that the end-effector has the supported firmware version. In the case of xMate CR models, the xPanel end-effector parameters should be correctly configured.

Parameter setting
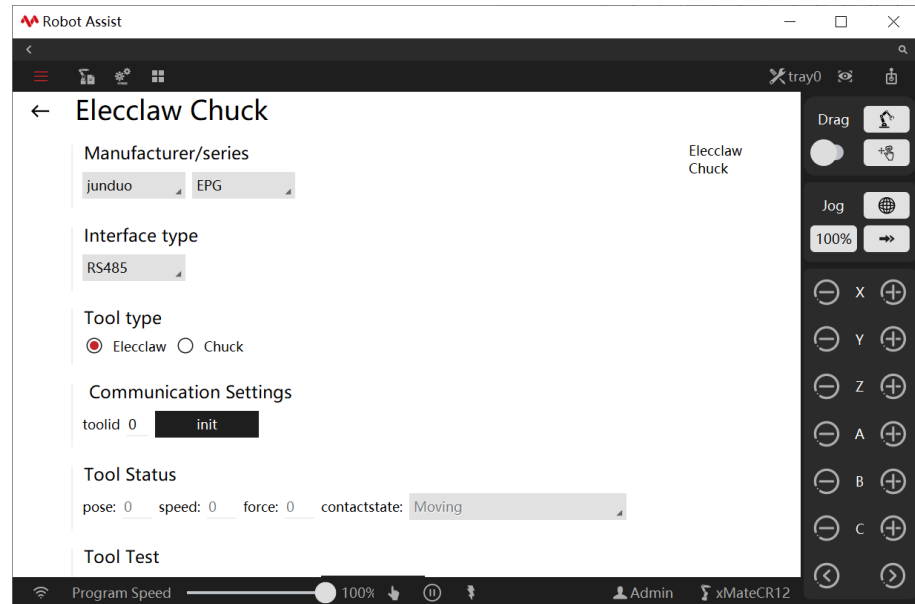
Currently, only Jodell Robotics with EPG series is supported through RS485 protocol.

Electric gripper and suction cup are supported. This interface is mainly used to test the hardware connection and communication of electric gripper and suction cup and whether the two tools function properly. The test interface of the electric gripper is shown above.

Initialization: to test an electric gripper, first enter the ID of the electric gripper and click the "Initialization" button. If the electric gripper performs automatic detection and prompts successful initialization , the electric gripper's hardware connection and communication are sound, and the user can proceed to the next step or use it.

Move to: After initialization, click the "Move to" button and control the electric gripper to move to the specified position in the specified velocity and force. If the electric gripper reaches the specified position or encounters objects and reaches the specified force, it will stop the motion, and the electric gripper's contact detection status will also be displayed on the test interface.

Parameter	Value/Description
Tool ID	Enter the ID of the electric gripper. This ID is the electric gripper ID set in the Jodell Robotics debugging software
Tool position	Set the position of the gripper, range 0-255
Tool velocity	Set the velocity of the gripper, range 0-255
Tool torque	Set the torque of the gripper, range 0-255

The suction cup configuration interface is shown below.

Initialization: to test a suction cup, first enter the ID of the suction cup and click the "Initialization" button. If the software prompts successful initialization, the suction cup's hardware connection and communication are sound, and the user can proceed to the next step or use it.

Setup: After initialization, the suction cup parameters should be set as needed. When all parameters are entered, click the "Setup" button to test the suction cup.

Parameter	Value/Description
Channel selection	The suction cup supports two channels. The user can choose the effectiveness of the two channels at will.
Minimum vacuum	Set the target vacuum level of the suction cup. The suction cup stops working when the inside vacuum level reaches this value

Maximum Vacuum	Set the target vacuum level of the suction cup. The suction cup starts working when the inside vacuum level is greater than this value
Timeout period	Times out when the minimum vacuum level specified is not reached in the specified time
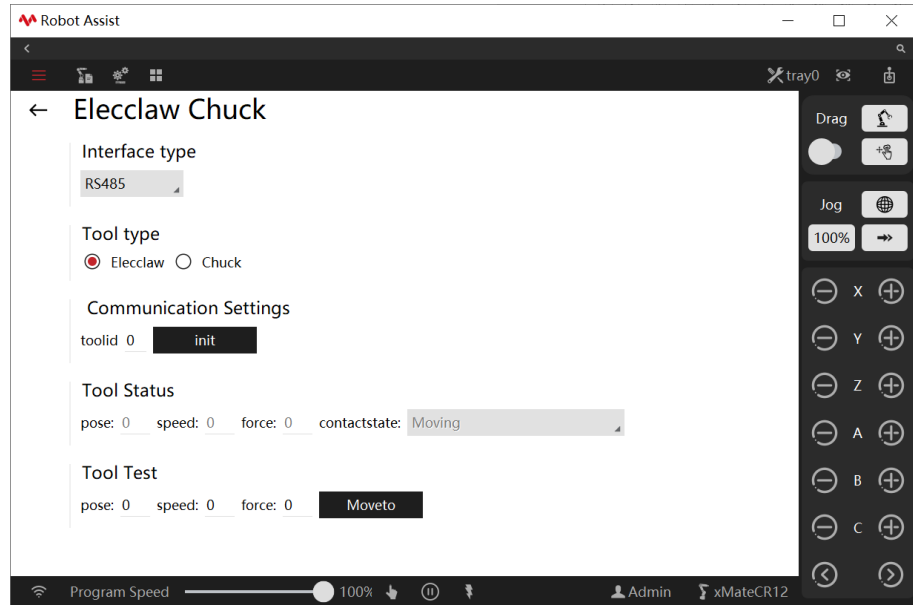
7.3.9 RCI setting

Explanation	
	RCI is an external control interface, and the RCI communication setting is required before use.

Parameter setting	
	The IP address of the user PC should be filled in before turning on the Enabling switch. If the user PC is directly connected to the robot via network cable, the IP address of the user PC should be in the same network segment as the IP of the robot; if the user PC is connected to the robot via wireless or router, the user PC should be in the same LAN as the robot. The port number is set to 1337 by default.
	The packet loss threshold is in percentage, which represents the packet loss rate during RCI communication. For example, when the packet loss threshold is set to 10, it means that the packet loss rate during RCI usage should not exceed 10%. The packet loss threshold is recommended to be set between 10-20.

Turn on RCI	
	After the communication setting is completed, turn on the Enabling switch and press the Save button to activate RCI.
	After using RCI, turn off the Enabling switch and press the Save button to disable RCI.
	Refer to the RCI User Manual for detailed RCI usage and routines.
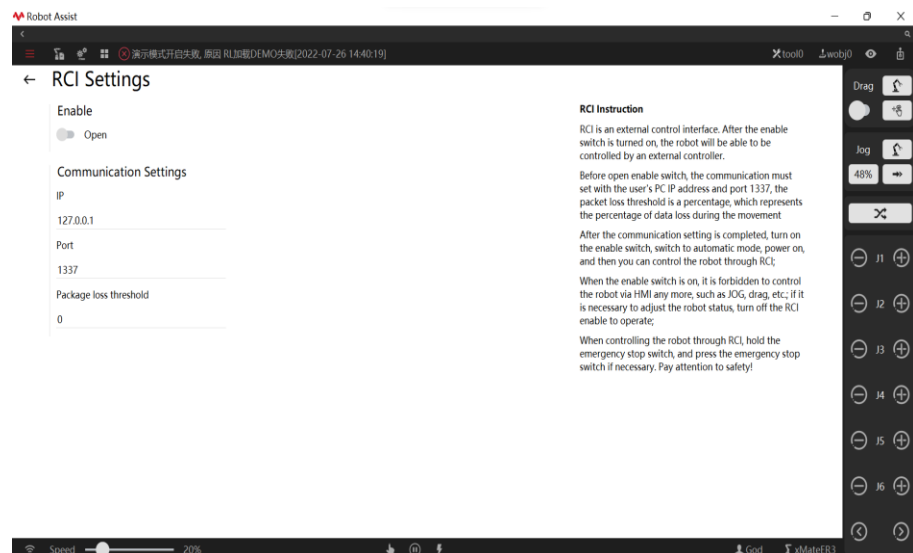
7.4 Process kit

7.4.1 Laser welding

Refer to *User Manual for Laser Welding Process Kit*

7.4.2 Plating line tracking

Refer to *Operation Manual for Plating Line Tracking*

7.5 Authorization

7.5.1 EtherCAT Authorization

Explanation

An authorization code is used to authorize the EtherCAT communication.

> **i** Notes
>
> The robot cannot be powered on if the authorization code expires or the authorization fails.

8 Menu module

8.1 Diagnosis

Explanation

The xCore system provides a detailed log of operations that can be used to trace the operation of the robot and identify the cause of the failure.
Filter the generated logs and view other operations using the log management interface.

8.1.1 Teach pendant log

Explanation
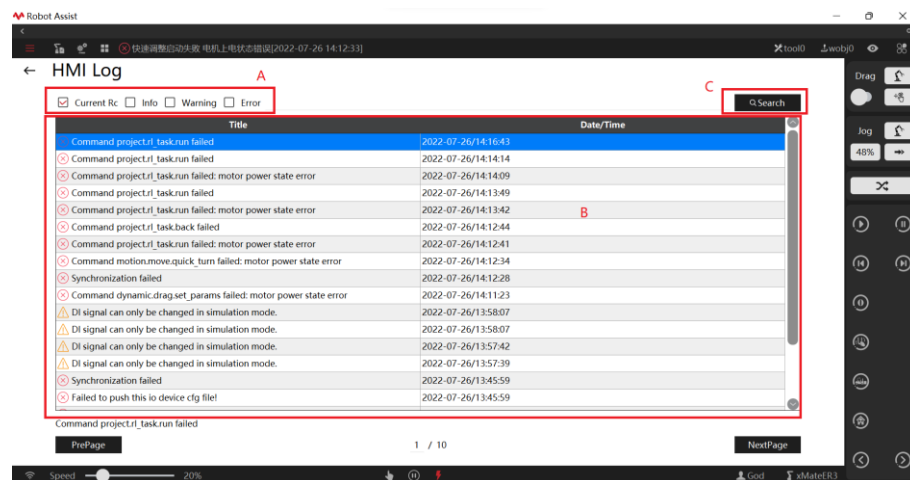
A	Filtering criteria area, in which the user can choose to view only the controller logs of the current controller or the ones connected to the HMI, as well as select the log level for further filtering
B	Log display page, where log title and generation time are displayed. The user can press Previous/Next to switch between the previous and next pages
C	Search area, where the user can search logs with keywords

8.1.2 Controller logs

Explanation

A	Filtering criteria area, in which the user can select log level for further filtering
B	Log display page, where basic information such as log number, title, generation time, and content is displayed. The user can press Previous/Next to switch between the previous and next pages
C	Search area, where the user can search logs with keywords

8.1.3 Log timeline
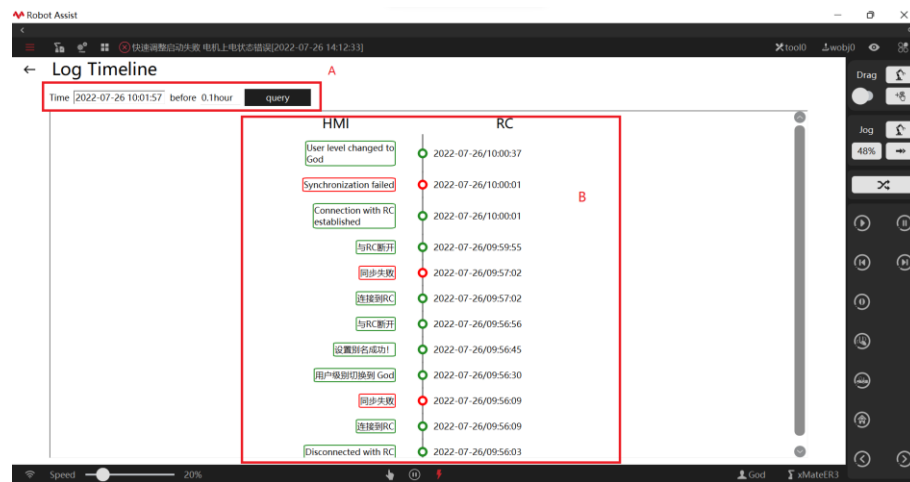
A	Search area, where the user can search logs with a specific time range
B	Log display page. HMI logs are displayed on the left and RC logs on the right

8.1.4 Internal logs

Explanation

When a customer encounters a problem on-site, technical support can determine the cause of the problem based on the log files.

8.1.5 Advanced options

Explanation

The Advanced Options interface is used to assist developers with the diagnosis of the servo, ECAT, and other equipment, and enable real-time thread alarm and monitoring, etc. Since enabling diagnostic function will increase the workload of the controller, only turn it on in actual production when necessary.
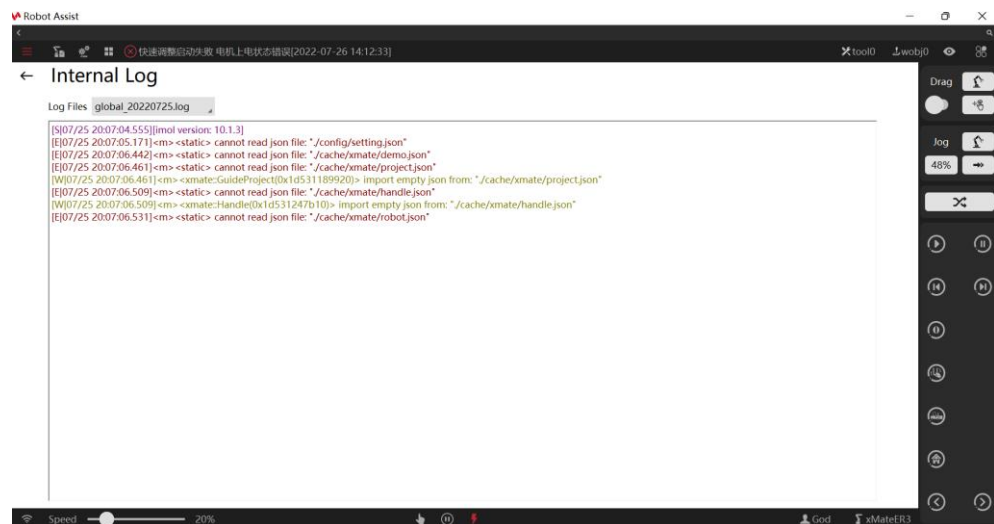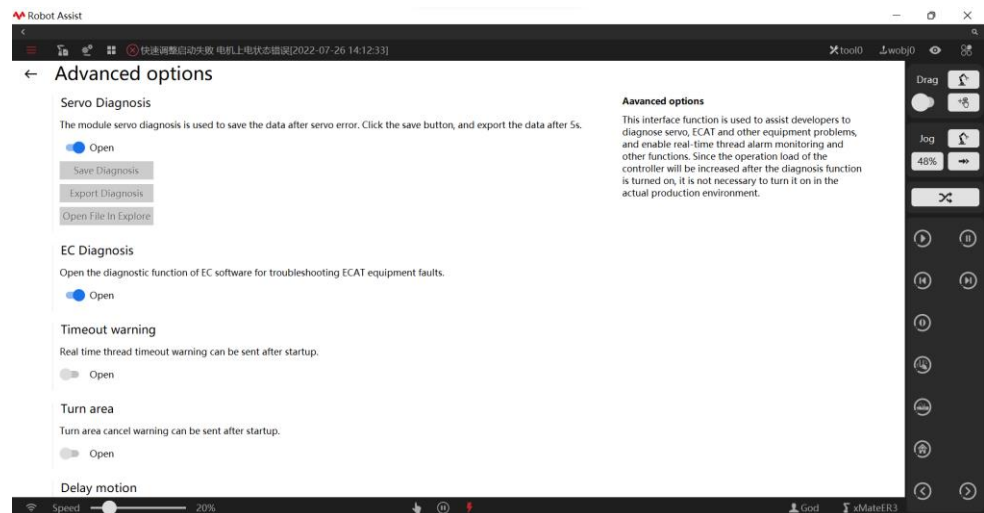
Servo diagnosis	The servo diagnostic module is used to save the data errors in the servo. Click the Save button. The diagnostic data can be exported after 5s.
EC diagnosis	The EC software diagnosis function can be used to assist in troubleshooting ECAT devices.
Timeout warning	Send real-time thread timeout warnings after enabled.

8.1.6 Error recovery

Explanation

When the robot reports errors, take the remedies suggested in the error details to recover from errors. Contact ROKAE for after-sales service if an unrecoverable error occurs.
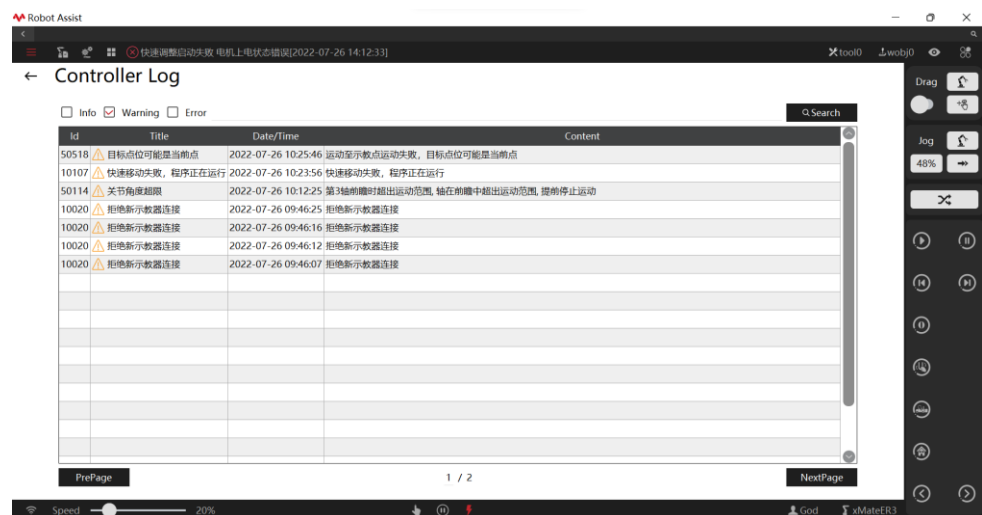
8.2 Help

Explanation

The Software Upgrade interface provides such functions as controller upgrade and backup, erase all configurations, and factory reset.
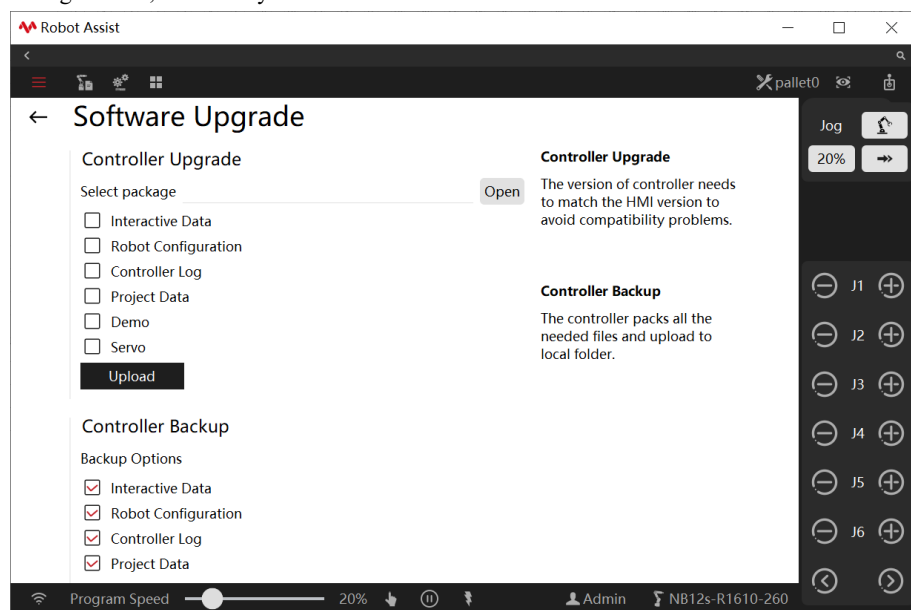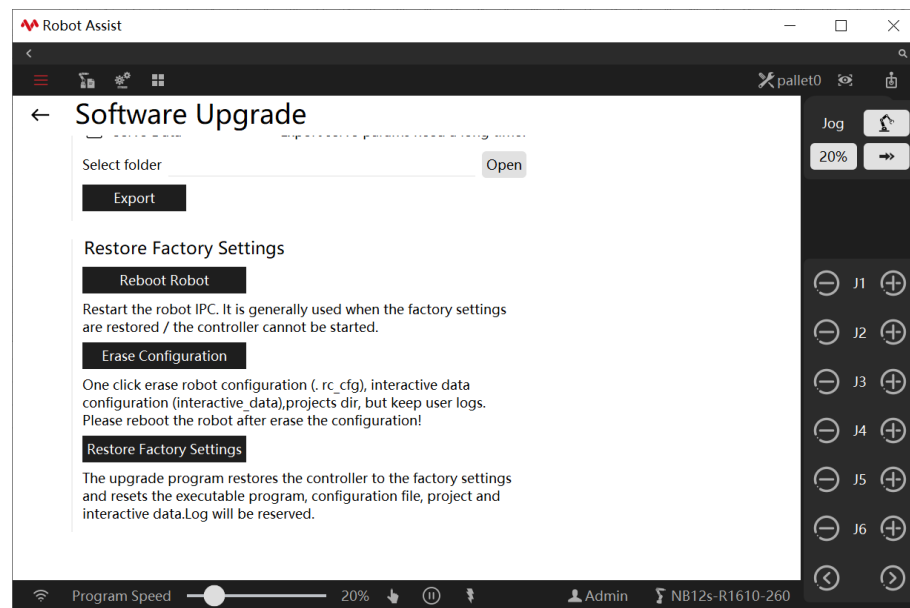
Controller upgrade: used to **upload upgrade package** and **restore data**. Two file formats are supported: encrypted file and unencrypted package. After uploading, the prompt "Uploaded successfully" will appear on the interface. Follow the pop-up prompt to restart the controller.

Data restoration: Select the data package for restoration through the controller upgrade, check the data for restoration, and click Upload. Follow the pop-up prompt to restart the controller.

Controller backup: used to store **backup data** of the controller. Select the files for backup, click Open to select the backup directory, and then click Export. An encrypted file is then exported.

HMI upgrade (for xPad2 only): Upgrade of the xPad HMI software. **Open:** Select the HMI upgrade zip file in the USB drive directory. **Upgrade:** Click the "Upgrade" button to start the HMI upgrade. After the HMI upgrade is completed, the HMI software will start automatically and the HMI upgrade is finished.

Restart robot: Restart the IPC system, and the upgrade service connection needs to be established for this operation.

Erase all configurations: Erase robot configuration files, custom configurations, user's project files, etc. with one click. However, the operation logs of the control system will be retained. After clicking this button, the user needs to manually restart the robot. The upgrade service connection needs to be established for this operation.

Factory reset: Restore the control system to its factory default state. The control system configuration files and the user's project files will be reset. However, the operation logs of the control system will be retained. The upgrade service connection needs to be established for this operation.

To upgrade to the latest version of the controller, please contact us and request an installation package of the latest controller version and the HMI software package.

- ➢ Download to local the installation package of the latest controller version and the HMI software package.
- ➢ Open the upgraded HMI and connect to the controller and the upgrade service.
- ➢ Select Custom Configuration, Robot Configuration, Controller Log, and Project Data in the Backup option, and export the backup after selecting the local folder directory for backup.
- ➢ Do not select Custom Configuration, Robot Configuration, Controller Log, Project Data Demo Case, or Servo in the Controller Upgrade option. Choose the controller installation package downloaded locally. The upgrade options will be configured based on the installation package. Then, click to upload.
- ➢ After a successful upload, the HMI will prompt to restart the controller. The controller will be successfully upgraded after the restart.

> **Notes**
>
> 1. To prevent data loss during the upgrade, we recommend backup the controller beforehand.
> 2. Do not select Custom Configuration, Robot Configuration, Controller Log, Project Data, Demo Case, or Servo during a software upgrade, otherwise, information such as current robot zero position, dynamics model, and project data will be overwritten.
> 3. Clicking the Erase all configurations or Factory reset button will reset the robot's key operation configuration files and delete user-defined project data. Please double-check before performing the operation!

> **Notes**
>
> When the HMI version does not match the controller version, HMI will display an instant log message in the top status bar, which reads "Version mismatch. Recommend HMI version: [xxx]".

8.3 Demos

8.3.1 Seven-axis redundant motion

Explanation

Common motions of the seven-axis redundant manipulator, including arc, straight line, turning zone, and null-space self-motion.

	Operation	Description
1	Use admin to log in to the system and switch to the demo interface.	
2	Select the feature for demonstration in the Demo list on the left.	
3	Click the mode switching button on the bottom status bar and switch to Auto Mode.	
4	Click the Power-On button on the bottom status bar to power on the robot.	
5	Click Play Demo in the upper right corner.	
6	Click Stop Demo in the upper right corner after demonstration and then select another demo.	To adjust the Demo threshold (such as sensitivity in collision detection or stiffness in compliance demo) during a demonstration, click Stop Demo first and replay the demo after adjustment.
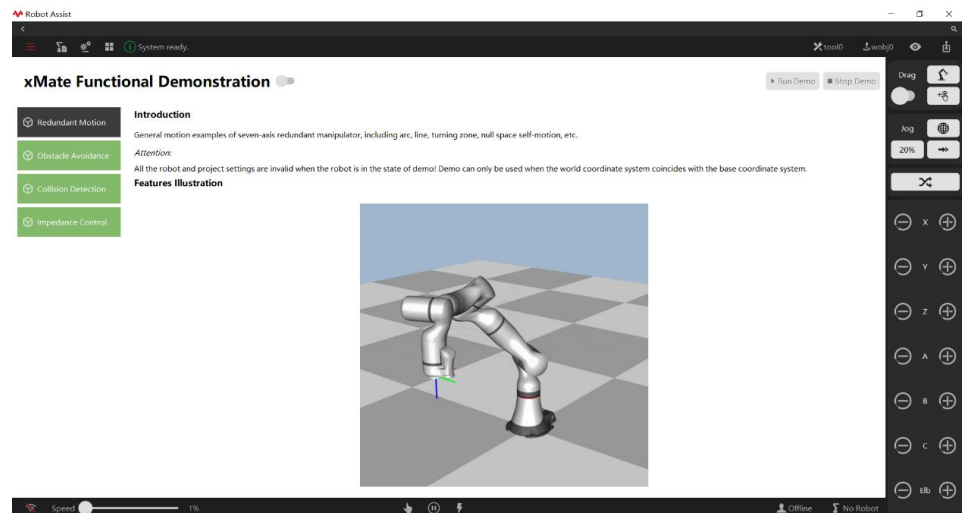
8.3.2 Obstacle avoidance

Explanation

When the manipulator enters a narrow and deep box, it will not interfere with the box structure by adjusting its orientation through null-space self-motion, which enables the robot to perform the pickup and delivery task successfully.
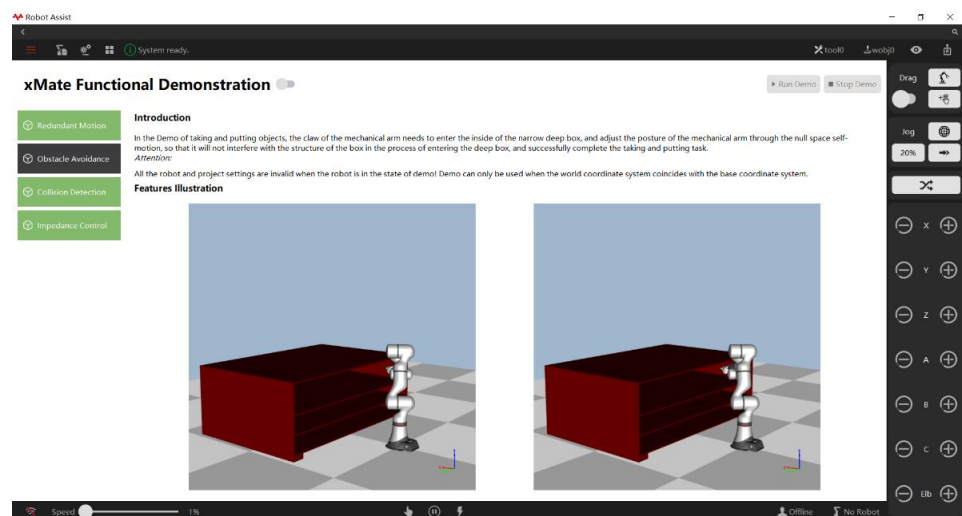
8.3.3 Collision detection

Explanation

Two settings are available in the collision detection demo: single-axis sensitivity setting; high, medium, and low sensitivity setting.

When the robot stops after detecting the collision, you can press on the robot for it to continue its normal operation.
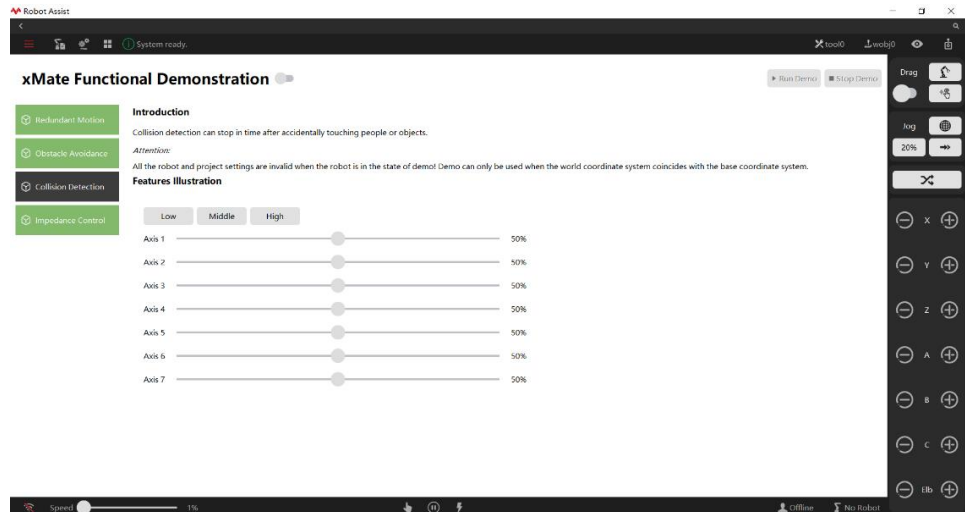
8.3.4 Compliance demo

Explanation

The compliance demo demonstrates the force control of xMate in Cartesian space with different stiffness. This function is suitable for grinding, polishing, deburring, and other processes with high grinding forces.

Two settings are available in the compliance demo: individual setting of end-effector translation, rotation, and elbow stiffness; combined setting of translational and rotational motion.
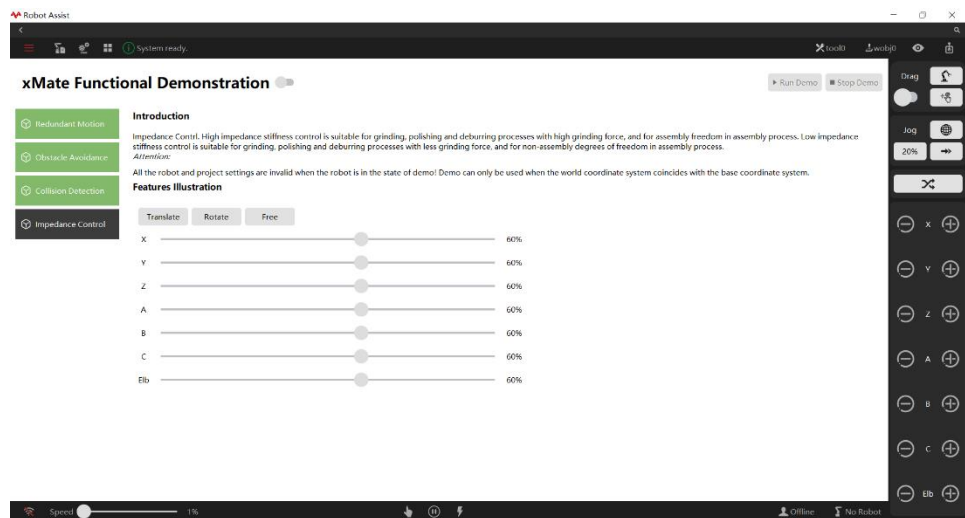
> ⚠️ Warning
>
> During the demonstration, all the robot configurations are invalidated. For example:
>
> 1、 The base frame of the robot coincides with the world frame by default.
>
> 2、 There is no load at the robot end-effector by default. Otherwise, the demonstration of collision detection or compliance will be affected.

9 Teach pendant options

9.1 Connection settings

Explanation

The connection interface is mainly used to detect and connect robots.

Search for available robots: Search for all robots in the same LAN (except direct connection). When the robot is connected, it will be displayed that the controller service and the update service are both connected.

Automatic reconnection: When the network between the robot and Robot Assist is disconnected, Robot Assist will try reconnection automatically and will stop trying after the preset reconnection time.

If **the robot can not be found** when Robot Assist and the robot **are in the same LAN,** or **the real-time position of the robot is not displayed on the 3D interface** after connecting the robot, click the blue text in the figure below or go to the Basic Settings interface and select the IP address assigned to the LAN in the Bound IP Address drop-down box.
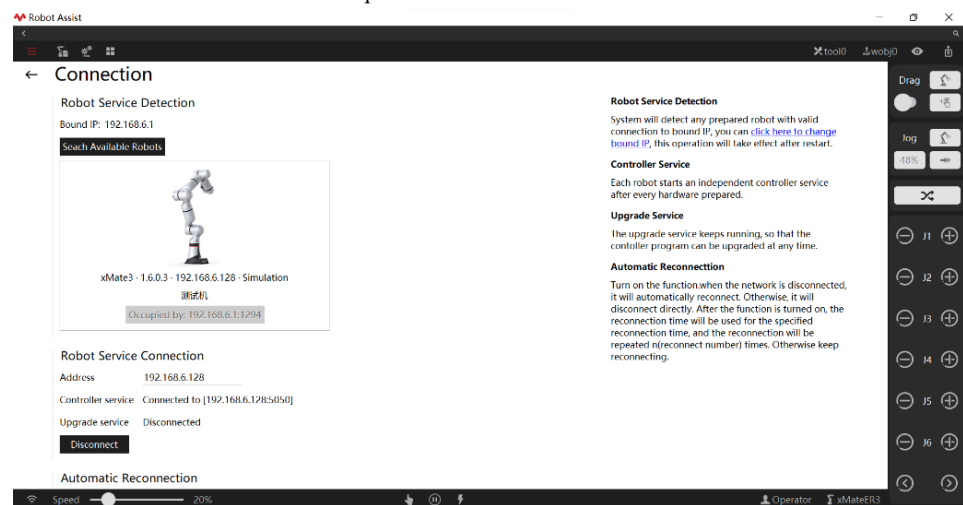
9.2 Basic settings

Explanation

Language settings: Chinese and English are optional.

Teach pendant IP setting (only for xPad2): Set the static IP address for the teach pendant connected to the robot.

Screenshot setting (only for xPad2): Take a screenshot of the teach pendant screen and save it in the teach pendant directory. The picture format is JPG.

Bind IP address: Set the network card for connection of Robot Assist and the robot.

Workspace directory: Set the folder to save project files.

Graphic performance: Set the time interval (unit: ms, 100ms max.) for 3D model refreshing.
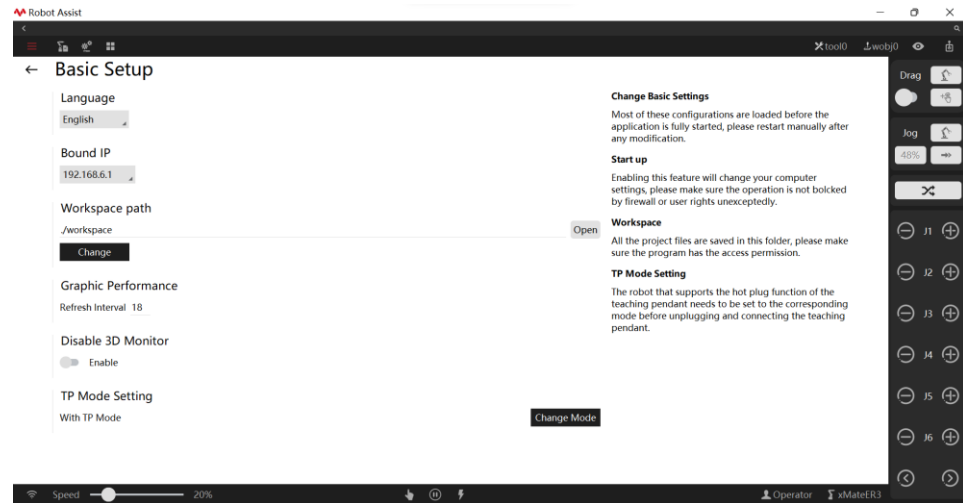
9.2.1 Multi-language log

Explanation	
	For xCore control system of version 1.7 and above, multi-language log is supported (only for "controller log").
	When HMI language changes, the controller will also change the language accordingly;
	When the HMI and the controller set different languages, the controller will switch to the language of the HMI when the connection is established.
	Restart is not needed when the controller switches language. But some detailed log information only takes effect after the switch, and the log generated before the switch may not be displayed in the corresponding language.

9.3 Appearance settings

Explanation	
	Themes: The default style adapts to Windows 7. Only the font is different between the two styles.
	Theme size: Set the interface font, control, icon, and layout size. Click Apply and restart to take effect.
	Mouse cursor switch (only for xPad2): Show and hide the mouse cursor. Takes effect immediately after checking the selection box.
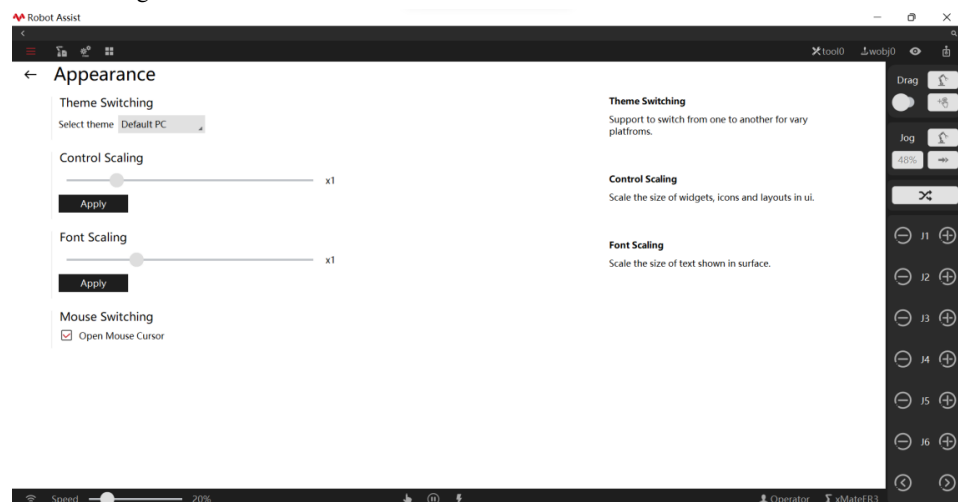
9.4 File manager

Explanation

The file manager interface is used to quickly open folders in the Robot Assist software package. This interface is only available for PC-based software.

Cache folder: used to store the cache of Robot Assist.

Log folder: used to open the folder with Robot Assist logs. The logs in the folder are consistent with the internal log on the diagnostic interface. Click to open the folder for an individual backup.

Workspace folder: used to quickly open the folder with the robot project files.
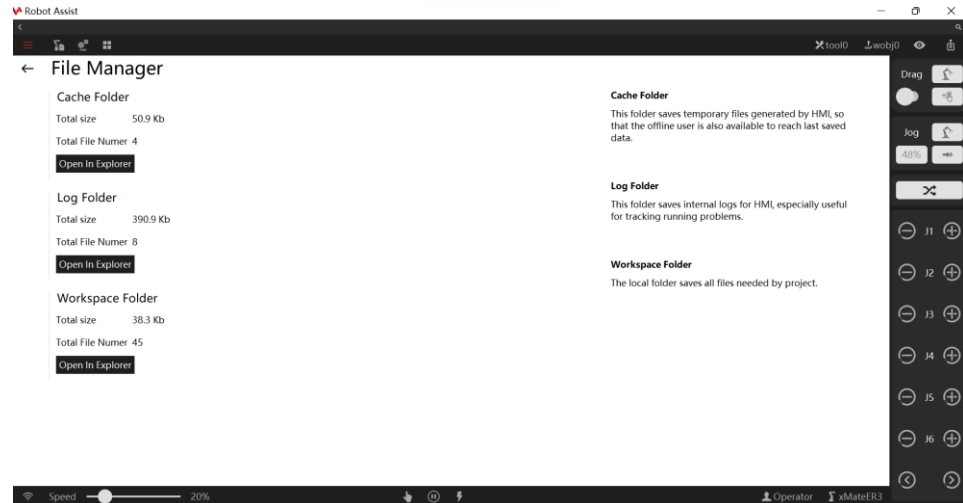
10 Robot Motion Foundation

10.1 Frame

Explanation

The motion of the robot contains information such as position, velocity, acceleration, etc. This information would be of practical significance by specifying the reference system. Therefore, before Jog starts, we need to understand the frames used by the robot.

In addition, defining and using an appropriate frame helps simplify the programming process and improve the use efficiency of the robot.

Frame

The xCore system adopts a rectangular frame (i.e. the Cartesian frame, hereinafter collectively referred to as Cartesian Frame) to describe the position and orientation in a three-dimensional space. The frames currently used in the xCore system is shown below:
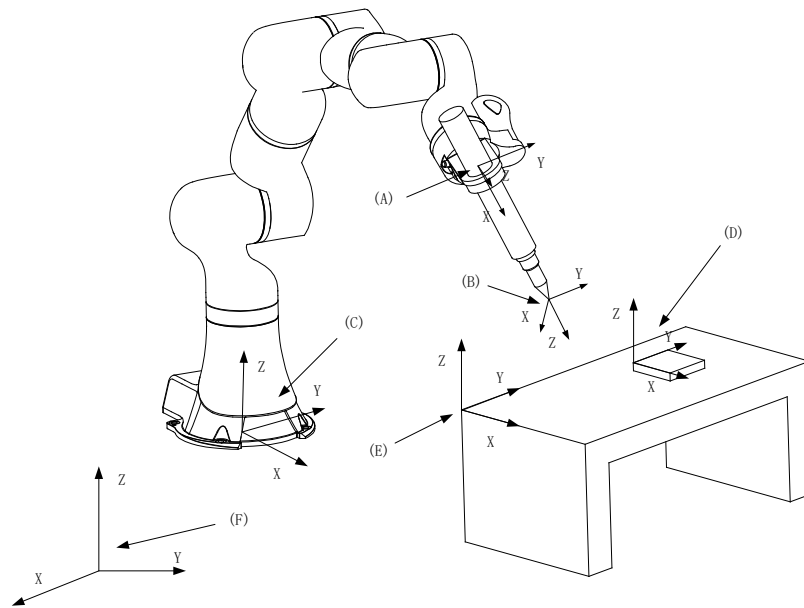
A. Flange Frame: It is defined at the center of the robot's end-effector flange but with no practical meaning. It only serves as a reference when defining the tool/work object frame.

B. Tool Frame: It is defined on the tool. The robot programming position refers to the position of the tool frame. For further information on the tool frame, refer to the Tools.

C. Base Frame: It is defined at the center of the robot base and is used to determine the robot's position.

D. Work Object Frame: It is defined on the work objects. A well-defined work object frame can greatly reduce the programming complexity and improve program reusability. For more information on the work object frame, please refer to the Work object.

E. User Frame: It is used as a reference frame when defining the work object frame, and it cannot be used separately.

F. World Frame: This frame does not have a specific position. When there is only one robot, the frame can be considered to be at the center of the robot base, which coincides with the base frame. When there are multiple robots or external devices that need coordinated motion, the world frame can provide a unique reference system for these devices. The specific position can also be arbitrarily specified on the premise that the base frame of other devices can be conveniently calibrated.

For the dependencies between different frames, please refer to variables "tool" and "wobj".

10.2 Robot singularity

Explanation

Under normal circumstances, the robot can use up to 8 different joint configurations to reach the pose in the same working space. For details, please refer to the introduction of the confdata variable. However, there are still a few special poses in the robot's working space that the robot can arrive at using a myriad of different joint configurations. Such poses are called singularities. Singularities may cause problems to the control system when calculating joint angles based on spatial position.

In general, the xMate robot features the following types of singularities:

1. Axis 2 singularity
2. Axis 4 singularity
3. Axis 6 singularity
4. Wrist singularity

There is no singularity problem when the robot performs joint motion.

When the robot performs a Cartesian space trajectory near the singularity, some joints may be very fast. In order to not exceed the maximum joint velocity, the speed of the end-effector trajectory will be automatically reduced.

Axis 2 singularity

When the angle of Axis 2 is 0°, the robot experiences the Axis 2 singularity:
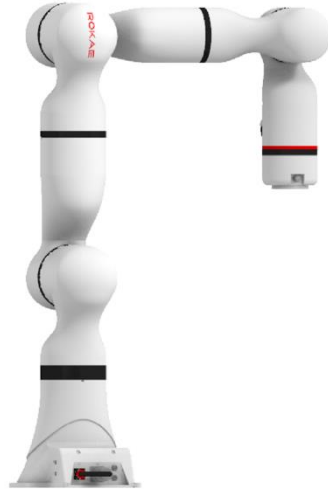
At this point, the robot cannot distinguish between the angles of Axis 1 and Axis 3 when calculating inverse kinematics.

Axis 4 singularity

When the angle of Axis 4 is 0°, the robot experiences the singularity, and the pose is called the extended position:
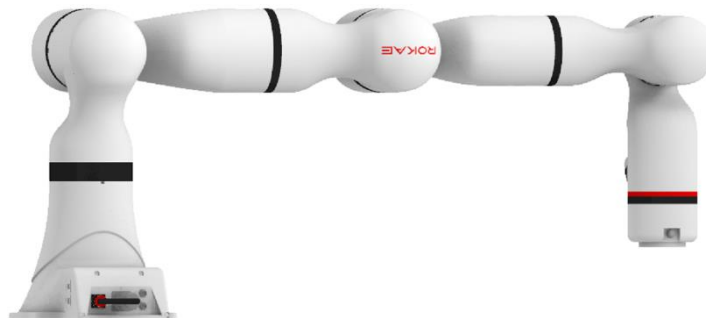
In this pose, the robot is restricted to move in the direction parallel to Axis 3 or 5. This singularity often appears when the robot is moving to the boundaries of the working space.

This singularity causes the robot to lose one degree of freedom at the root of the wrist (the root of the wrist is unable to perform axial motion along the arm). In this case, the Axis 3 and Axis 5 positions cannot be obtained when calculating inverse kinematics.

Axis 6 singularity

When the angle of Axis 6 is 0°, the robot experiences the Axis 6 singularity.

At this point, the robot cannot distinguish between the angles of Axis 5 and Axis 7 when calculating inverse kinematics.

Wrist singularity

When the robot's wrist center is right above Axis 1, the robot experiences the wrist singularity.

At this point, the robot cannot precisely calculate the angle of Axis 1 when calculating inverse kinematics.

10.2.1 Turning zone

Explanation

For the manipulator, its motion is usually executed sequentially according to multiple trajectories specified by the user. However, different trajectories specified by the user are usually not smoothly connected to each other, so there are various "spikes". The presence of these "spikes" forces the robot to stop at the end of a trajectory before executing the next trajectory. To enable continuous motion between trajectories, it is necessary to eliminate such "spikes" and generate turning zones to smoothly connect different trajectories specified by the user. See the following figure:
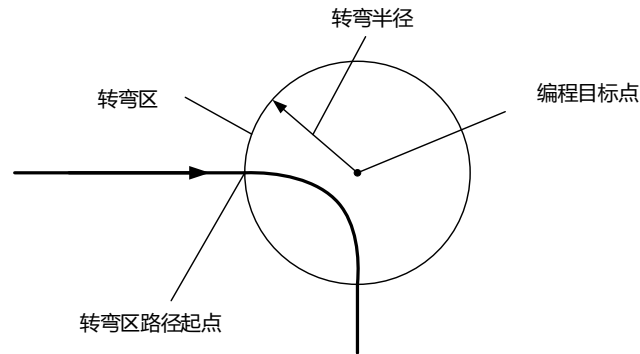
In addition, when the manipulator is moving in a joint space, it also moves along different trajectories. The difference is that the trajectory is now no longer defined by the Cartesian space pose but by the angle of each axis. This means that there should be turning zones when the robot moves in the joint space.

For specific parameter settings of the turning zone, please refer to the variable "zone".

10.2.2 Lookahead mechanism

Explanation

Lookahead is: the control system handles the commands after the command the robot is currently executing in advance during the movement. The introduction of the lookahead mechanism can be advantageous in the following aspects:

➢ Obtain the speed of the front trajectory, the acceleration requirements, and the constraints of the robot itself, so as to plan the optimal control strategy;

➢ Plan the turning trajectory of the turning zone according to the settings of the programmed turning zone;

➢ Acquire an abnormal state near the soft limit/boundary, singularities, etc., so that it can be processed in advance;

The lookahead mechanism cannot be closed manually. The system automatically looks ahead when running the program. You can use the Program Pointer to view the lookahead position.

Some RL commands will interrupt the lookahead. When the compiler encounters such a command, it will stop compiling until the robot executes the compilation of the corresponding command. Only Print command, logical judgment command, and user-defined functions do not interrupt the lookahead mechanism, and all other functions will interrupt the lookahead mechanism.

10.3 Robot force control

10.3.1 Introduction to force control

The robot force control is a process of interaction between the robot end-effector and forces in the external environment. In non-contact robot motion control, only the position control process (velocity and accuracy) is considered. When there is contact with the environment, pure position control requires very high accuracy of the robot and the environment to avoid damaging the robot and the environment due to contact forces caused by position deviation. Unlike pure position control, robot force control adopts a force/torque feedback loop when interacting with the environment. The loop is used to change the motion characteristics of the robot, which enables dynamic interaction with the external environment. When there is deviation or uncertainty between the robot and the external environment, the force control will intelligently adjust the preset position trajectory to eliminate the internal force caused by the position deviation, thus ensure a smooth and safe interaction process.

10.3.2 Impedance control

Compared with traditional industrial robots, xMate is equipped with joint torque sensors, which enable it to sense joint torque precisely. The joint torque information allows xMate to achieve force control through impedance; this enables compliant interactive behavior of the robot. The interaction between the robot and the environment is like a virtual spring stiffness and damping system. At this point, the robot is sensitive to external forces, which can cause the robot to deviate from a predetermined trajectory. When the external force disappears, the robot can rebound to some extent.
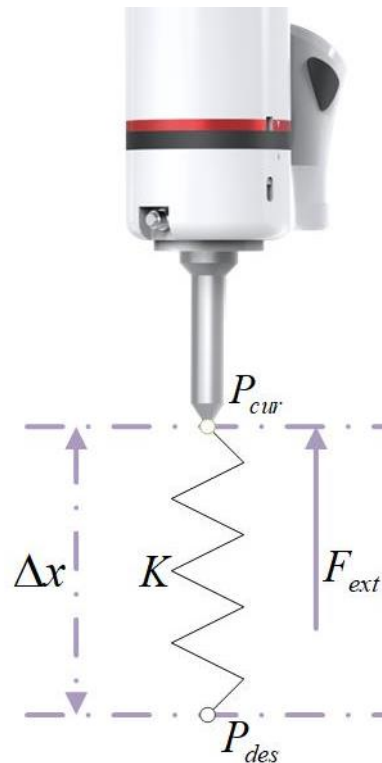
In the process of impedance motion, the actual position of the robot will deviate from the desired position when affected by the external forces in the environment. The deviation depends on the impedance stiffness and the external forces, and it can be calculated through the ratio between the external force and the impedance stiffness. As shown above, the impedance stiffness is set to K in the impedance control mode. Affected by external force F_{ext}, the robot's current position P_{cur} will deviate from the desired position P_{des}, and the position deviation is Δx. The impedance force caused by this deviation and the external force will eventually reach an equilibrium.

The impedance stiffness in each direction can be set individually, and the impedance force in each direction is the product of the impedance stiffness and the position deviation in this direction. The impedance force in each direction adds to the total impedance force. In the figure below, the robot's current position P_{cur} deviates from the desired position P_{des} affected by external forces in the impedance mode. The deviations in the X and Y directions are Δx and Δy, the impedance stiffnesses are K_x and K_y, and the impedance forces are F_x and F_y, respectively. The total impedance force $F = F_x + F_y$.
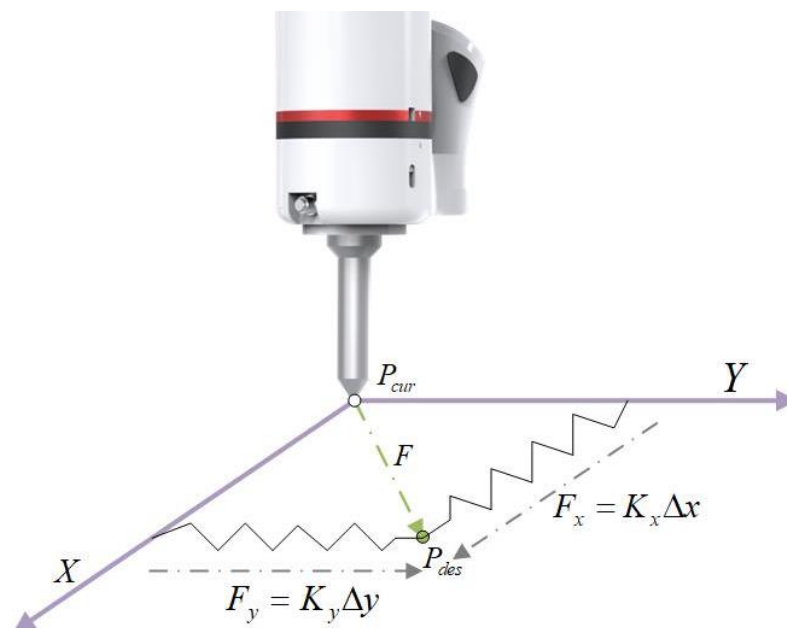
10.3.3 Overlay

When assembling work objects, humans would feel the change in force. If an obstruction (the work object is stuck) is detected, humans will try shaking to ensure a smooth installation. Force control allows xMate to do the same thing, i.e. Motion search. xMate supports sine overlay rotating around an axis and Lissajous overlay within a plane. Overlay is an additional motion added to the specified motions. It allows the robot to shake, which enables it to better overcome obstacles during the assembly process. Below is a sine overlay:

1、 Desired trajectory

2、 Actual trajectory (desired trajectory + overlay)

3、 Overlay amplitude

4、 Overlay period
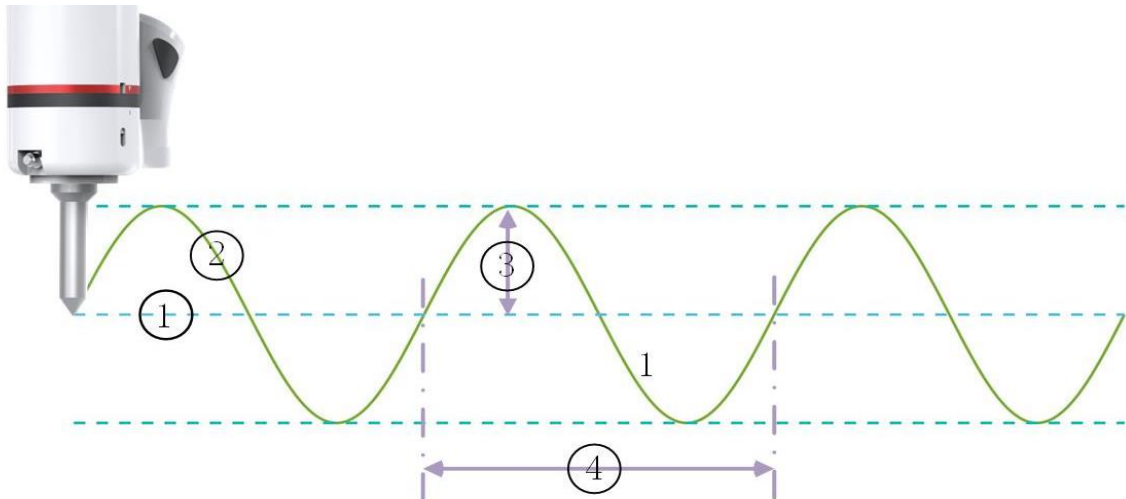
Lissajous overlay means a sine overlay in two perpendicular directions within the plan, and the frequencies of the two overlays are often proportional. For example, below shows the Lissajous overlay in the XY plane, where the frequency ratio of x- and y-direction overlay are 2:1. The center point P_{start} is the desired pose, X_{amp} is the amplitude of the x-direction overlay, and Y_{amp} is the amplitude of the y-direction overlay.
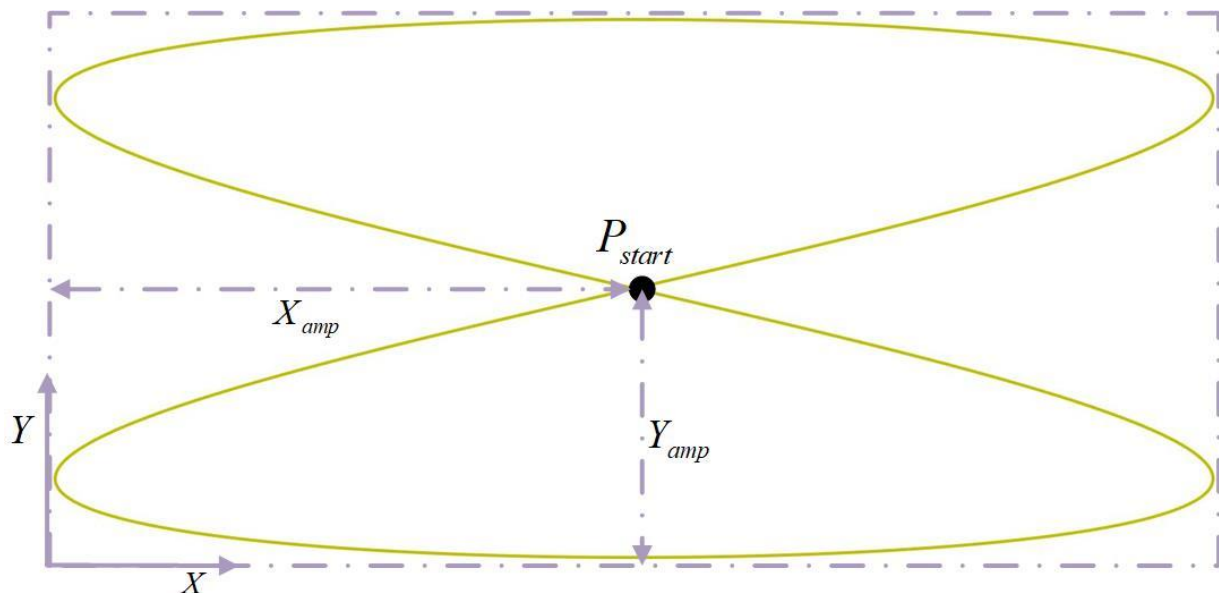
10.3.4 Applications

The application scenarios of force control for industrial robots fall into two categories: **constant force tracking** and **force-controlled assembly**.

1)   Constant force tracking

Below shows a constant force tracking scenario. The robot ensures a constant contact force F_{des} with the surface, while the robot can conform to the surface curve. Main applications include grinding and deburring.
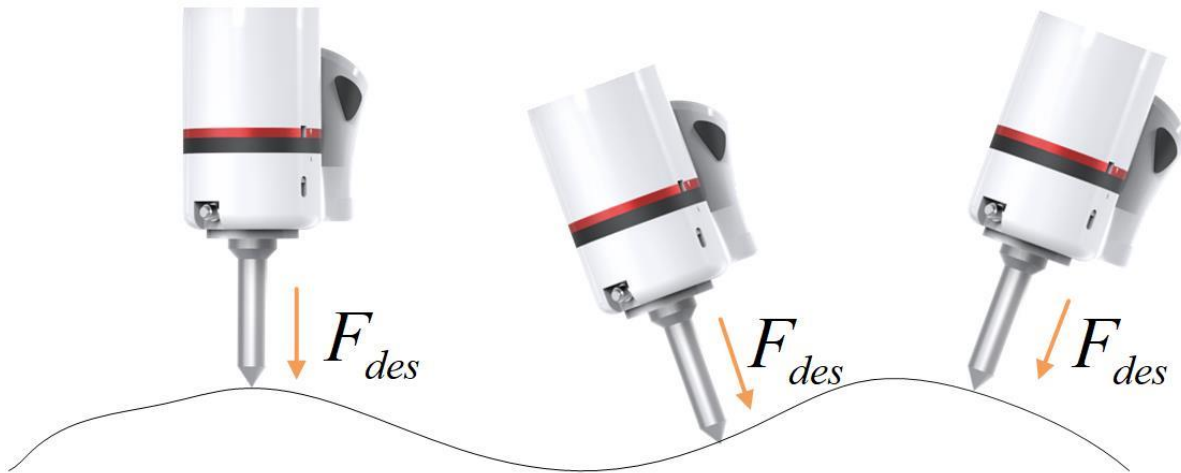
Example program for constant force grinding: The robot is set to Cartesian impedance mode. The impedance stiffness and load information are set, and force control is enabled. The work object is pressed onto the grinding surface through a desired force in the z-direction. The observed force in the z-direction is monitored during the pressing process, and when the observed force exceeds a certain threshold, the tool is considered to have contacted the surface. At this time, a desired trajectory in the grinding direction is applied. The robot maintains a constant force during the motion, thus allowing constant force grinding.

```
VAR POSE T_POSE = PE:{0, 0, 0},{1, 0, 0, 0} //Tool frame
VAR POSE W_POSE = PE:{0, 0, 0},{1, 0, 0, 0} //Work object frame
FcInit T_POSE, W_POSE, 0 //Force control initialization. 0 means force control frame is set as the base frame
SetControlType 1 //Set Cartesian impedance as the impedance mode. 0: joint impedance 1: Cartesian impedance
SetCartCtrlStiffVec 500, 500, 0, 100, 100, 100 //Set the Cartesian impedance stiffness. The first three are translation
stiffness (0 ~ 1500), and the last three are rotation stiffness (0 ~ 100)
SetCartNsStiff 2.0 //Set null-space stiffness (0~4)
SetLoad 0.82,0,0,0.041,0,0,0 //Set load information
FcStart //Enable force control
SetCartForceDes 0, 0, -15, 0, 0, 0 //Set desired force in the Cartesian space. Apply -15N force in the z direction
FcCondForce -100, 100, -100, 100, -100, 10, true, 20.0 //Cartesian space force monitoring. Apply 10N force in the z
direction to trigger
FcCondWaitWhile //Enable the monitoring conditions set before
MoveL p0,v800,z50,tool0 //Motion command (desired trajectory)
FcStop //Disable force control
```

2) **Assembly**

If pure position control is used during the assembly, the robot may easily collide with the work object due to position and modeling errors, which can cause damage to the work object or the robot.

But with force control, the robot will try to overlay (shake) to overcome the obstruction when it senses an external force over the limit (work object jamming), thus allowing successful work object installation. As shown below, the position control on the left results in a collision during assembly, while the force control on the right pushes the robot into the assembly hole through the desired force F_{des}, and the jam is prevented through overlay $F_{overlay}$.
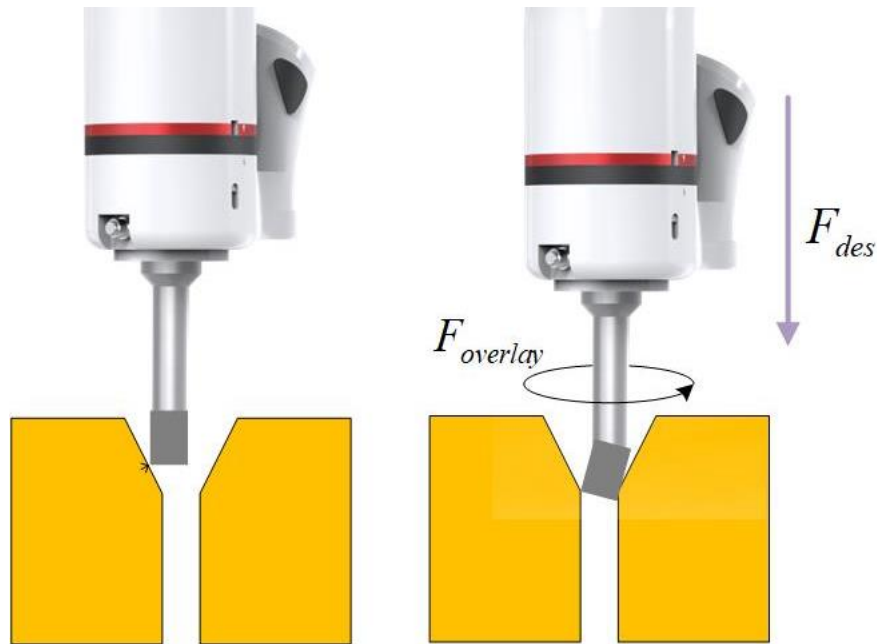
Example program for force-controlled assembly: The robot is set to Cartesian impedance mode. The impedance stiffness and load information are set, and force control is enabled. The work object is pressed into the mounting hole by applying the desired force in the z-direction. The observed force in the z-direction is monitored during the press-in process, and when the observed force exceeds a certain threshold, the tool is considered to be stuck. At the time, the preset Lissajous overlay is executed to ensure that the work object is successfully pressed in. The position in the z-direction is monitored to determine whether the work object has been successfully installed.

```
VAR POSE T_POSE = PE:{0, 0, 0},{1, 0, 0, 0} //Tool frame
VAR POSE W_POSE = PE:{0, 0, 0},{1, 0, 0, 0} //Work object frame
FcInit T_POSE, W_POSE, 0 //Force control initialization
SetControlType 1 //Set Cartesian impedance as the impedance mode. 0: joint impedance 1: Cartesian impedance
SetCartCtrlStiffVec 500, 500, 0, 100, 100, 100 //Set the impedance stiffness. The first three are translation stiffness (0 ~ 1500), and the last three are rotation stiffness (0 ~ 100)
SetCartNsStiff 2.0 //Set null-space stiffness (0~4)
SetLoad 0.82,0,0,0.041,0,0,0 //Set load information
SetLissajousOverlay 0, 5, 5, 5, 5, 0 //set XY plane for Lissajous overlay, 5N, 5Hz,5N, 5Hz, 0rad
FcStart //Enable force control
SetCartForceDes0, 0, -15, 0, 0, 0 //Set desired force in the Cartesian space. There is a -15N desired force in the z direction
FcCondForce -100, 100, -100, 100, -100, 10, true, 20.0 //Cartesian space force monitoring. Apply 10N force in the z direction to trigger
FcCondWaitWhile //Enable the monitoring conditions set before
StartOverlay //Start overlay
VAR fcboxvol box1 = fcbv:{-1000.0, 1000.0, -1000.0, 1000.0, 250.0, 500.0} //Define a box area
FcCondPosBox F_POSE, box1, true, 20.0 //Box area monitoring. Triggered when the robot is out of the box area
FcCondWaitWhile //Enable the monitoring conditions set before
FcStop //Disable force control
```

11 Programming and Debugging

11.1 Programming preparation

Programming Tool

You can use Robot Assist and Rokae Studio for programming. Robot Assist is suitable for online program modification, such as position and path variables. Rokae Studio is suitable for offline programming and simulation, including project design, model selection, trajectory generation, trajectory optimization, simulation debugging, and code generation.

For details on how to use Rokae Studio for offline programming, please refer to *Rokae Studio User Manual*.

Define tool, payload, and work object

The tool, payload, and work object should be defined before programming. The default tool is tool0 and default work object is wobj0. Then you can define more required objects at any time.

Define frames

Confirm that the base frame is correctly set during robot installation.

You can define the tool frame and work object frame if needed before programming. Corresponding frames should be defined when adding more objects later.

11.2 Project

11.2.1 Project introduction

Project overview

In the xCore controller, a project refers to the management collection of programs, tasks, and other objects that control the operation of the robot. It is responsible for storing all the necessary information needed for the robot to work, including:

➢ Task list;
➢ Variable list;
➢ Point position list;
➢ Path list;
➢ IO signal list;
➢ User frame list;
➢ Tool frame list;
➢ Work object frame list;
➢ Predefined parameters;
➢ Vision system;

Open project

Click in the upper left corner to enter the Project interface;
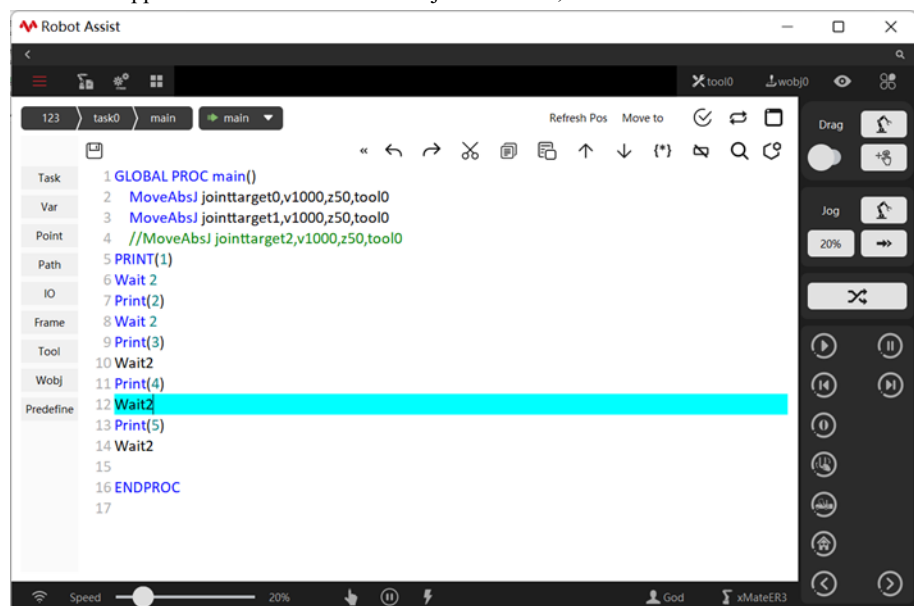
11.2.2 Project configuration

Explanation

The Project Configuration interface is used for the relevant configuration of the current project, including:

➢ Sync projects between Robot Assist and the controller;
➢ Switch between projects;
➢ Import/export projects;
➢ Create, modify, and delete projects.

Sync projects between RobotAssist and the controller

Once the connection to the controller is established, local projects will be loaded and updated immediately to stay consistent with those in the controller. Changes to important local project files will be immediately synchronized to the controller to ensure proper robot functions, including tools, work objects, and user frames. As RL codes are flexible, they will only be automatically pushed to the controller during debugging. If unfinished work should be saved, click the Push to Controller button to manually push the project to the controller.

Switch between projects

Click the drop-down menu below the selected project to display all projects. Select the desired project and click Reload to switch to the project.

Create project

Click ⊞ to create a new project. The project name can only be a collection of letters, numbers, and underscores "_".

After clicking the Create button, you will enter the New Project Wizard interface for easy creation and import of related configurations. The default task for a new project is Task 0.

Modify project

Click ✎ to modify the current project.

Delete project

Click 🗑 to delete the current project.
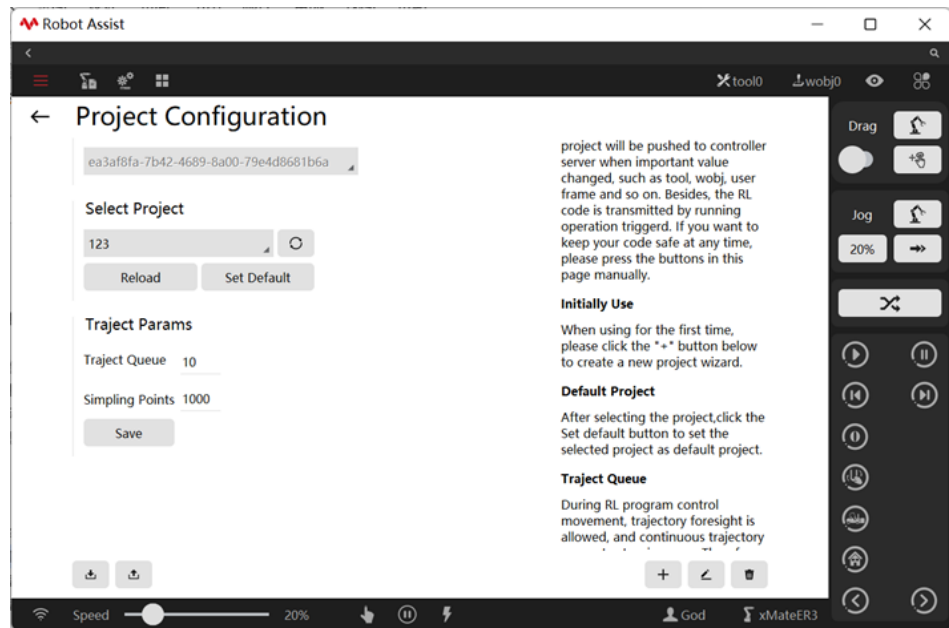
⚠ Warning
Files cannot be recovered once deleted!

11.2.3 Task list

11.2.3.1 What is Multitasking?

Explanation

The multi-task function is for running multiple robot programs at the same time; this function is particularly useful in the following conditions:

➤ Monitor continuously one certain signal even if the main program stops operating. It is similar to the background PLC function, but its response speed is much lower.

➤ The robot can send or receive various information when performing the main motion program, without any restriction on the actuating logic of the main program.

➤ Receive some inputs through the teach pendant while working.

➤ Control and activate/deactivate external device.

11.2.3.2 Task list

Overview

The xCore system provides an interface for managing parallel tasks. The interface displays the attributes of parallel tasks. And, the control logic of each task is implemented in Task. In the interface, users can create, set, and delete tasks.
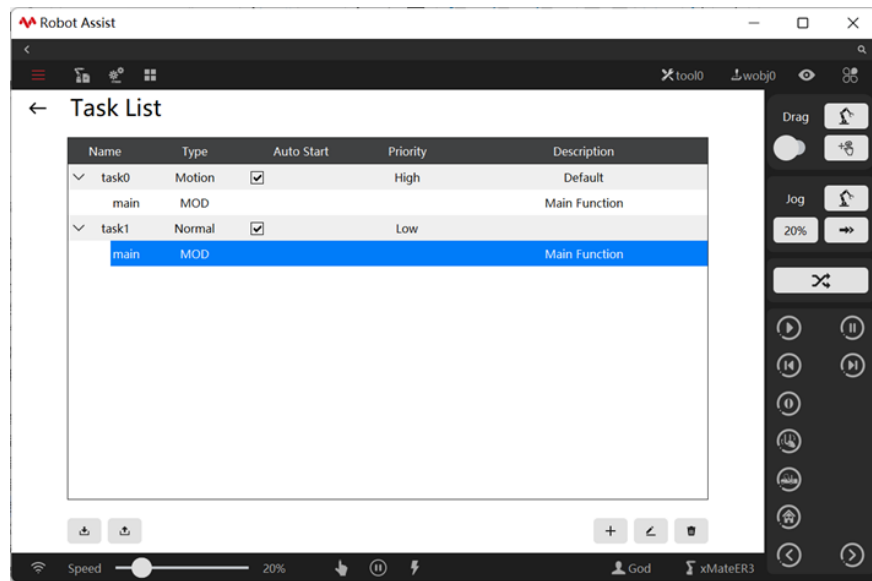
11.2.3.3 New task

Task attribute

Task attribute	Description
Task name	The task name must be unique among all tasks, it shall only be composed by numbers, letters, or underscores. Its initial character shall not be a number and the maximum length of the task name is 30 characters.
Task type	Motion tasks refer to those that allow RL commands to control the robot motion. There shall be only one motion task.
Autostart	It is used along with the Production mode. When selected, the program starts to re-execute when the system is restarted. Normally, it will not be stopped by the teach pendant or emergency stop.
Priority	Set the task priorities
Create file	When the main function generation is checked, the main function will be generated automatically after task creation. The same applies to other functions.

New task

At least one project shall be ensured in the resource manager when a new task is created.
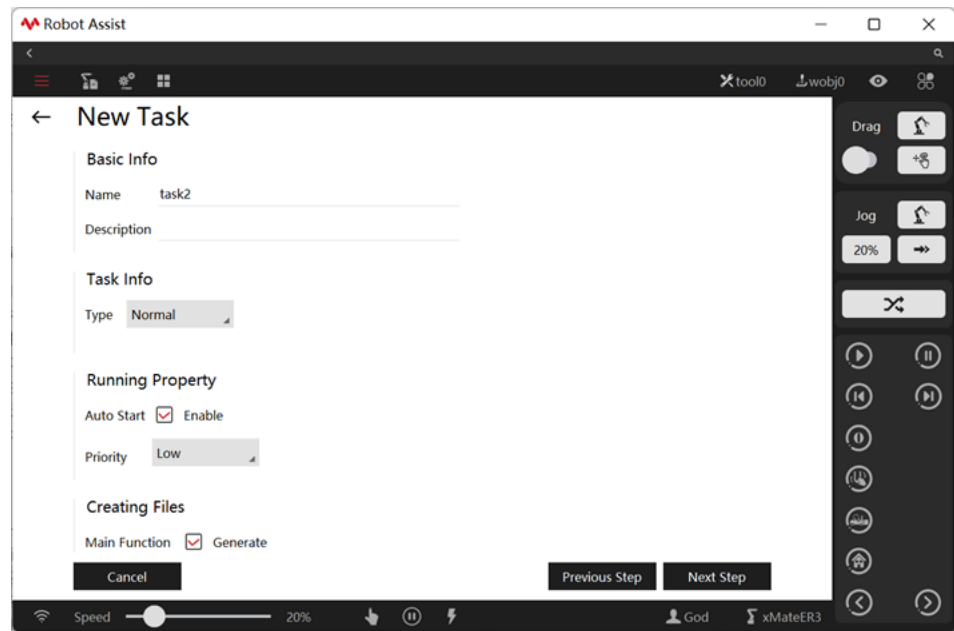
Use restrictions

> ➤ Support up to 10 tasks.
> ➤ There shall be one motion task at most.
> ➤ Changes in the task type, task entry function, and whether a motion task take effect immediately.

11.2.3.4 Starting and running tasks

Explanation

Click ![task0] on the RL Code interface to select the task. Use the Start/Stop button or external signal to control the start/stop of the selected task in case of manual or automatic enabling.
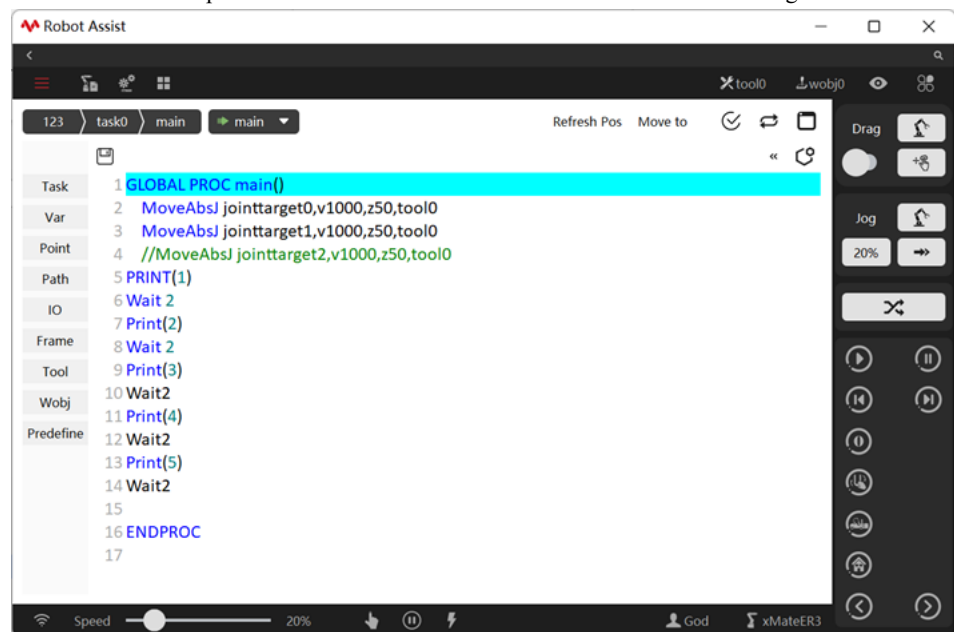
Use restrictions

> ➤ Generally, a background task will run cyclically. If a task does not contain any wait commands, the background task may consume too much computing resources, causing the controller to be

unable to handle other tasks.

➤ The scopes of variable VARS and the constant CONST are limited to their respective tasks, but the Global-level PERS variable is a global variable.

➤ When PPToMain is executing, all non-running tasks execute PPToMain.

➤ When there are tasks running, it is forbidden to modify the contents in the Task List interface.

11.2.3.5 Intertask Communication

Explanation

The intertask communication supports two methods: PERS variable and interruption.

Intertask communication by PERS variable

➤ Global-level PERS variables with the same name shall be defined in all task projects that required communication and the type, dimension of variable data shall be identical.

➤ PERS variable shall be used to control task execution and data transmission where necessary.

Intertask coordination of execution sequence by interrupt

➤ Define interrupt and corresponding interrupt handler function in the task that requires waiting.

➤ Set interrupt trigger signal at the right place of the task being awaited.

Use restrictions

➤ Simply specify the initial value for the PERS variable in one of the tasks. If you have specified an initial value for the same PERS variable in multiple tasks, the initial value defined in the first running task will be used.

➤ When a task waits for another task by means of the PERS variable and the WaitUntil or WHILE command, it is necessary to pay attention to coordinate with wait command (greater than 0.1s) to avoid the program quickly executing the empty judgment command, and thus occupying too much system resources.

11.2.4 List of variables

11.2.4.1 Variables

11.2.4.1.1 Basic concept

Variable naming rules

Variable names in the RL language can consist of letters, underscores, and numbers. However, variable names cannot be the same as system keywords. See Keywords pre-definition for RL system keywords.

In addition, there are the following precautions:

1. In the same module, GLOBAL and LOCAL level variables with duplicated names are not allowed;

2. In different modules, GLOBAL variables with duplicated names are not allowed;

3. In different modules, LOCAL variables with duplicated names are allowed;

4. In the same module, no variable (GLOBAL, LOCAL, excluding ROUTINE) is allowed to have a name that conflicts with functions in this module;

5. In different modules, no GLOBAL level function and variable naming conflicts are allowed;

> **ℹ Notes**
>
> When a variable name contains two characters only, please pay attention to not name the second character "h" or "b", otherwise, the variables may be converted to hexadecimal or binary.
>
> For more information, please refer to the Number system conversion.

Variable scope

The RL language system defines three scopes:

1. GLOBAL, visible to all modules in the current project, can be declared in the module declaration area;

2. LOCAL, visible only to the current module, can be declared in the module declaration area;

3. Functions (ROUTINE), visible only within the current function, can only be declared within the function body, and the scope type (GLOBAL or LOCAL) is not allowed to be specified when the scope variable is declared;

> **i** Notes
>
> Function (ROUTINE) scope applies to variables only, not to custom functions.

Storage type

Each variable is divided into variable (VAR), persistent variable (PERS), and constant (CONST), depending on whether it can be modified during program execution.

➢ VAR (Variable), a variable that can be reassigned during a program run;

➢ CONST (Constant Variable), which cannot keep up with the change of the new value in the process of operation, must be determined at the beginning;

➢ PERS (Persistent Variable), a continuous variable, during the execution of a program, if the value of the variable type changes, the variable will be automatically amended from the initial value to the current value, thus achieving the effect of "Persistent" storage.

> **i** Notes
>
> 1. Even if the value of a PERS type variable is changed while the program is running, the initial value displayed in the program editor declaration area is not immediately refreshed, and the initial value displayed in the program editor declaration area is updated to the latest value only when the program reloads.
> 2. The latest value of the PERS variable can be viewed at any time in the "variable management" interface, whether the program is running or not.

Keywords pre-definition

The following are reserved keywords (case insensitive) that are predefined for the RL language:
Module, EndModule, Proc, EndProc, Func, EndFunc, TRAP, ENDTRAP, SetDO, DO_ALL, SetGO, SetAO, WaitDI, Wait, WaitUntil, WaitWObj, WBID, Q, P, J, V, W, T, S, L, CA, DURA, IGNORELEFT, EJ, 1J, FCBV, FCCV, FCOL, FCXYZ, FCCART, PE, PER, TCP, ORI, EXJ, CFG, PDIS, JDIS, MoveAbsJ, MoveJ, MoveL, MoveC, MoveT, LOCAL, TASK, GLOBAL, VAR, CONST, PERS, INV, DOT, CROSS, sin, cos, tan, asin, cot, acos, atan, atan2, sinh, cosh, tanh, ln, log10, pow, exp, sqrt, ceil, floor, abs, rand, GetCurPos, Print, PrintToFile, ClkRead, TestAndSet, IF , Else, Endif, WHILE, ENDWHILE, for, from, to, endfor, Break, Continue, Del, Int, Double, Bool, String, BYTE, Robtarget, Speed, Zone, Tool, Wobj, Jointtarget, TriggData, Load, FCBoxVol, FCSphereVol, FCCylinderVol, FCXyzNum, FCCartNum, Pose, CLOCK, INTNUM, SYNCIDENT, TASKS, Call, Return, EXIT, Pause, StopMove, StartMove, StorePath, RestoPath, True, False, Interrupt, When, Offs, CalcJointT, CalcRobT, CRobT, RelTool, SocketCreate, SocketClose, SocketSendByte, SocketSendInt, SocketSendString, SocketReadString, SocketReadBit, SocketReadInt, SocketReadDouble, AccSet, MotionSup, TriggIO, TriggJ, TriggL, TriggC, On, Off, clock, intnum, userframe, pinf, ninf, FCFRAME_WORLD, FCFRAME_TOOL, FCFRAME_WOBJ, FCFRAME_PATH, FCPLANE_XY, FCPLANE_XZ, FCPLANE_YZ, FC_LINE_X, FC_LINE_Y,

FC_LINE_Z, FC_ROT_X, FC_ROT_Y, FC_ROT_Z, Offs, CalcJoinT, CalcRobT, CRobT, RelTool, \\Start, \\Time, ClkReset, ClkStart, ClkStop, CONNECT, WITH, IDisable, IEnable, ISignalDI, \\Single, \\SingleSafe, WaitWobj, DropWobj, WobjIdentifier, WobjAngle, ActUnit, DeactUnit, INTNO, \\Exp, DoubleToStr, WaitSyncTask, FCAct, FCDeact, FCLoadID, FCCalib, FCSupvForce, FCSupvTorque, FCSupvPosBox, FCSupvPosSphere, FCSupvPosCylinder, FCSupvOrient, FCSupvOrient, FCSupvReoriSpeed, FCSupvTCPSpeed, FCCondForce, FCCondTorque, FCCondOrient, FCCondReoriSpeed, FCCondPosBox, FCCondPosCylinder, FCCondPosSphere, FCCondTCPSpeed, FCCondWaitWhile, FCRefLine, FCRefRot, FCRefSpiral, FCRefCircle, FCRefForce, FCRefTorque, FCRefStart, FCRefStop, FCSetSDPara

---

Number system conversion

The RL language supports direct entry of hexadecimal, binary, or values of scientific notation by adding a number system identifier to a number or letter.

Example 1

Add the "h" suffix to 0~9, a~f, or A~F. The RL compiler treats the corresponding number or letter as hexadecimal and converts it to decimal in the compiler, for example:

8h stands for 8 in hexadecimal and 8 in decimal;

bh stands for b in hexadecimal and 11 in decimal;

25h stands for 25 in hexadecimal and 37 in decimal;

Example 2

Add the "b" suffix to 0~9, a~f, and A~F. The RL compiler treats the corresponding number or letter as binary, for example:

1b stands for 1 in binary and 1 in decimal;

10b stands for 10 in binary and 2 in decimal;

1010b stands for 1010 in binary and 10 in decimal;

Example 3

Add "e±x" after the number to indicate that the number is multiplied by 10 to the x power. For example:

5e+20 represents $5×10^{20}$;

26e-15 represents $26×10^{-15}$;

112e-10 represents $112×10^{-10}$;

11.2.4.1.2 Variable declaration

Explanation

A declaration must be made before using the variable. The format of the variable declaration command is as follows:

SCOPE STORAGE TYPE varname [= value]

Among them:

1. SCOPE is for variable scope. Please refer to Variable Scope;

2. STORAGE is for variable storage type. Please refer to Storage Types;

3. TYPE is for variable type and can be a basic type or a special type. Please refer to Variable Type;

4. varname is the variable name. Please refer to Variable Naming Rules;

The content in square bracket [ ] is optional and can be either initialized or not when variables are declared. For variables that are not explicitly initialized when they are declared, the system automatically assigns different initial values as per the type of the variable. The default initial value may cause program execution problems in some cases. It is recommended to initialize each manually added variable.

Example

The followings are a few examples for variable declarations:

Example 1

VAR int counter = 8 //Declare the integer variable count and assign an initial value of 8

VAR double time = 2.5 //Declare floating-point variable time and assign an initial value of 2.5

VAR bool ifOpen = true //Declare the variable bool type ifOpen and assign the initial value to true

Example 2

In general, no duplicate names are allowed for variables:

VAR int counter = 8

VAR double counter = 2.5

The compiler will report an error at this time by prompting "Failed to add variable".

Example 3

However, global variables and local variables can have the same variable name:

VAR int counter = 1

GLOBAL int counter = 555

Although variables with different scopes allow duplicate names, it is not recommended to use duplicate variables in order to avoid confusion and misuse, unless the variables with duplicate names have special technological advantages.

ℹ Notes

Variables cannot be declared inside a block of while and other loop commands, otherwise, duplicate declarations are caused when this part of the code is repeatedly executed, resulting in a "Fail to add variable" error.

Please declare the variables outside the loop body.

Use restrictions

> The ROUTINE variable that declares the PERS storage type is not supported;
> When there is a duplicate name for variables or functions of different levels, the compiler will decide which variable to be used based on the priority of the scope. Variables with the highest priority order will be selected first, and those with lower priority order will be obscured and hidden. The priority of scopes is as follows:
> ■ When the variable names are duplicated, the priority of scopes is as follows: ROUTINE> LOCAL> GLOBAL;
> ■ When the function names are duplicated, the priority of scopes is as follows: LOCAL > GLOBAL;

11.2.4.1.3 User variable hold

Explanation

Create user variable "a" with hold in an RL project. This user variable is marked as a pers variable, then the value of this variable is held on the non-volatile storage media when RL stops, the robot restarts, shuts down, or powers off. When the robot powers on again or RL is running again, the value of variable a is restored to the value held. The initial value is assigned only when the variable is created for the first time or re-edited.

Hold is available for such user variable types

Int, byte, double, bool, string, robtarget, jointtarget, pose, speed, zone, fcboxvol, fcspherevol, fccylindervol, fcxyznum, fccartnum, torqueinfo, tool, wobj

User variable hold configuration

The variable hold is accessible on the RL project interface as shown in red box below:
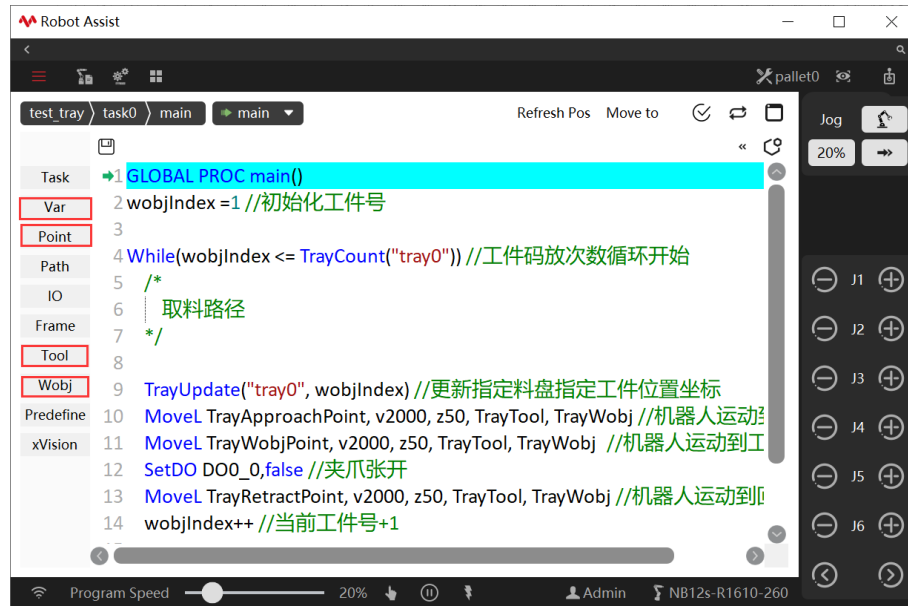
Click variable, point position, tool, or work object to create user variables of that type. All variables that support the hold property have a "persistent" drop-down box. Selecting true means that the variable is a hold variable, i.e., marked as a pers variable. For example, to create a pers variable of type int, configure it as follows: (and so on for other types)
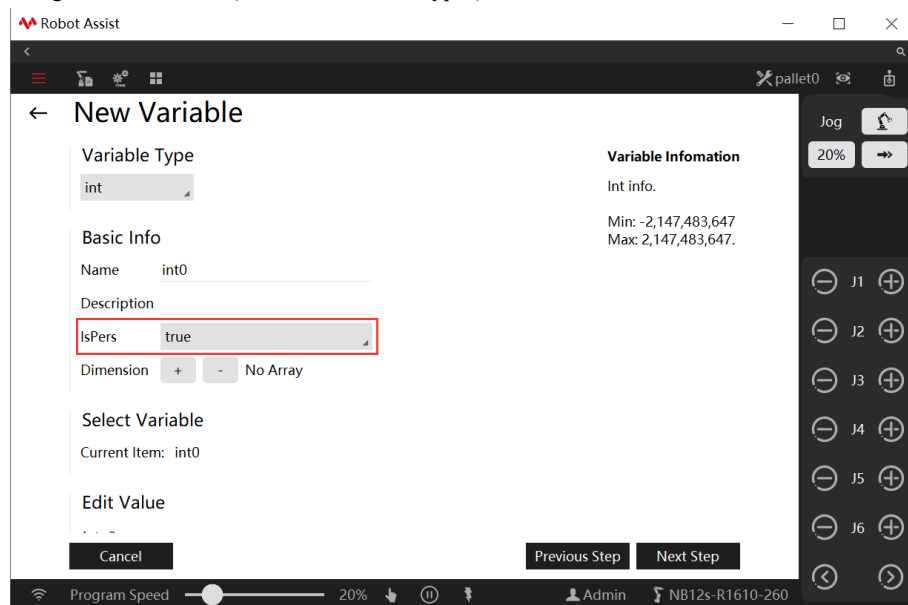
11.2.4.2 List of variables

Explanation

The variable management interface allows the creation, viewing, modification, and deletion of almost all variables in the robot system. The supported variable types include:

No.	Variable type	Description
1	System predefined variables	Variables that cannot be modified by users; it is used to store certain system parameters, such as tool0/wobj0.

2	User predefined variables	Variables that can be modified by users and used in multiple programs, such as user-calibrated tools, work objects, etc.
3	Program variables	Variables defined by the user in the program, which are generally used only in the current program and its subprograms. Program variables contain most of the types of variables supported by the system.

For some types of variables that have specifically defined steps, such as tool/wobj (defined and modified using the calibration interface), robtarget/jointtarget/speed/zone (defined and modified using an auxiliary programming interface). Although variables can be viewed and modified in the Variable View interface, it is still recommended to use the dedicated interface for modification for the sake of convenient operation and fewer errors. You are advised to only view variables in the variable management interface.

> **Notes**
>
> The variables that can be viewed and modified in the variable management interface are limited to the variables used in the currently loaded robot program,
> so the variables displayed will change after other programs are loaded.

Variable editing

If you need to add variables or modify an existing variable, you can click the "New" or "Modify" button to enter the variable editing page for operation.
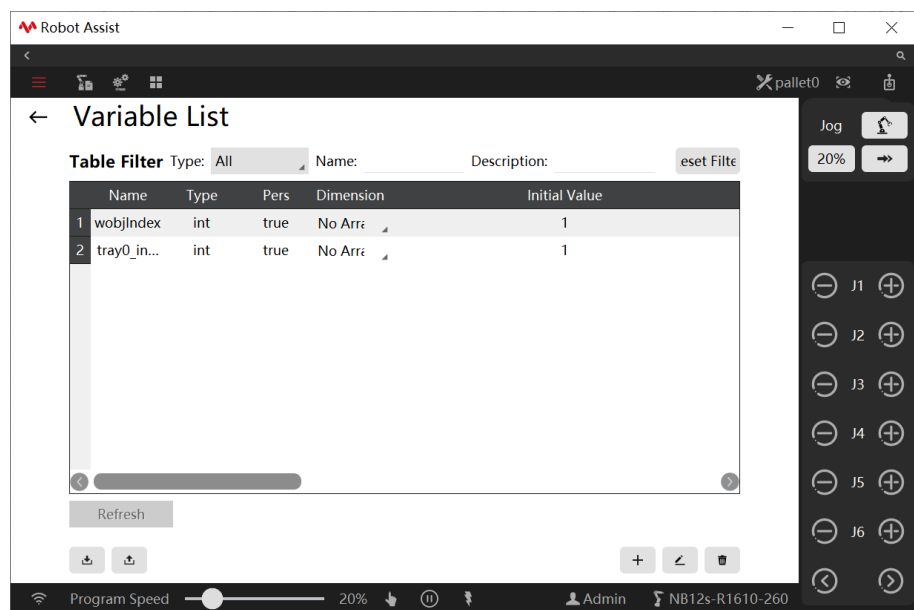
Variable type	Used to select variable types when creating a new variable. All supported types are listed in the sidebar on the left.
Variable name	The name of the variable to be inserted.
Array dimension	To create or modify arrays, supported up to 3D arrays.
Module name	The default is main.mod, or optionally stored in other mods.
Storage type	Choose between CONST, PERS, and VAR. For more information, please refer to the Variable declaration.
Scope	Choose between GLOBAL and LOCAL. For more information, please refer to the Variable declaration.

11.2.5 Point position list

Explanation

The xCore system provides an interface for the management of teaching point positions. The information of point positions used in the RL program needs to be configured in the point position list before they can be used in the program.

In both the point position editing interface and the point position list interface, the current pose can be used to update the teaching point position.

Add, modify, and delete point positions

The information of all point positions is configured on the point position list interface as shown below:
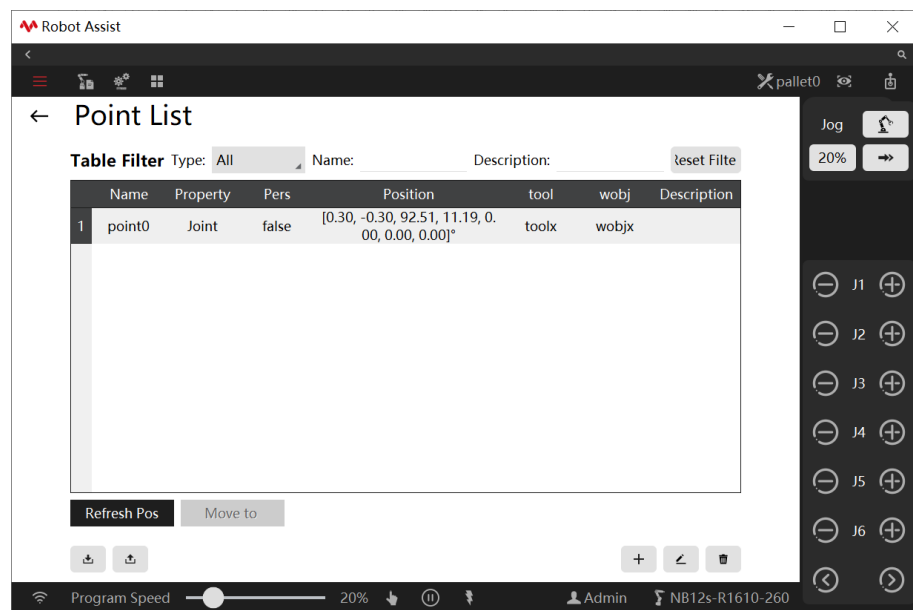
A	Name, which can be changed when performing the "New" or "Modify" operation.
B	Type, including joint space and Cartesian space.
C	Position: for joint space, displays the joint angle of seven axes for joint space; for Cartesian space, displays the xyz coordinate and manipulator angle in the base frame.
D	Description: Users can describe the point position, and the description can be changed when performing the "New" or "Modify" operation.

Edit point position

	Operation	Description
1	Use admin to log in to the system and switch to the point position list interface	
2	Click "+" in the bottom right corner to enter the point position creation guide interface.	You can also click ✎ to modify the point position or 🗑 to delete the point position.
3	Name the current point position in the name field.	
4	Add a description to the current point position in the description field.	Optional
5	Click to update the position.	Update the teaching point with the current pose.
6	Select Cartesian or joint space point position	This is used to update the point position manually. If the method in step 5 is used, this step can be omitted.
7	Manually enter the point position pose according to the point position's attributes.	

11.2.6 Path list

TBD

11.2.7 IO signal list

Explanation

In the xCore system, all common IO signals (including Profinet signals) can only be used in the programs after being configured on the control panel's I/O Signal List interface, except for the default signals.

signalxx type variables are used to store and access IO signals in the RL program. For details, refer to the RL section.

Add, Modify, and Delete IO

The configuration of all general-purpose IO signals is done on the relevant page of the control panel, as shown in the following figure:
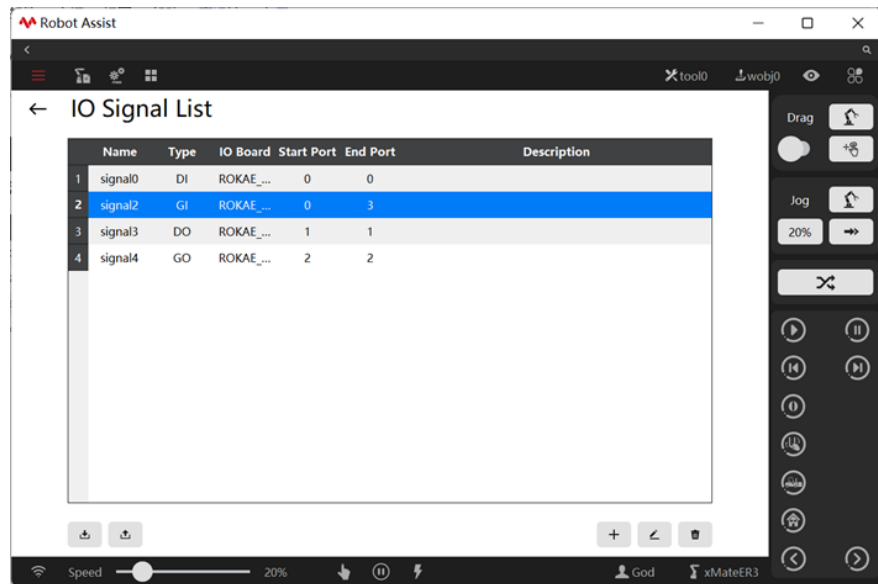
A	IO signal names that can be changed at the time you press "New" or "Modify".
B	The type of signals, including signaldi, signalgi, signaldo, signalgo, etc.
C	IO module number, which can be a standard IO module provided by the company, or the Profinet bus or Ethernet/IP bus
D	Address number, the physical address number corresponding to the IO credit mapping, starting from 0
E	Function button area, on which you can new, modify, and delete IO signals.

⚠️ Warning

If there is an error in the IO configuration, for example, when the mapped IO port exceeds the physical limits or if the port is repeatedly assigned, the control system will enter the SYS_ERR state and give an alarm message on the HMI at the time of starting. In this case, the user is only allowed to enter the system configuration interface, to correct the wrong configuration with no other operation allowed.

View IO

The configured general-purpose IO can be viewed on the status monitoring interface, and only the configured IO can be seen. The forced output or simulation input of the IO is supported.
General-purpose IO cannot be viewed in the variable management interface.

Use IO

For the input signal (DI/GI), the state of the input node can be read directly in the RL program using the variable name of the input signal.

Example 1

//Use the state of the digital input as a criterion for judgment
IF (di1 == true)
do something…
ENDIF
For the output signal (DO/GO), special commands SetDO and SetGO can be used in the RL. See the Explanation of each command for details.

Use restrictions

➢ User-defined IOs cannot be mapped to system outputs.

11.2.8 User frame list

Explanation

The user frame is used as a reference frame when defining the work object frame, and it cannot be used separately.

Calibration of the user frame

The method for calibrating the user frame is the three-point method. Its operating steps are the same as the three-point method for calibration of the tool frame.

Before calibrating the user frame, the user needs to calibrate a tool and then use the TCP of the tool to calibrate the user frame. For more convenient operation, it is recommended to use tools with tips.

	Operation	Description
1	Use admin to log in to the system and switch to the user frame calibration interface.	
2	Name the user frame to be calibrated in the name field.	
3	Select the "Calibrate now" in the pose calibration option.	Manual input is allowed if the user frame is known in advance. Calibration is not mandatory, and the user frame defaults to the world frame.
4	Jog the robot, make TCP of the calibrated tool point at the origin of the desired work object frame, and then click "Confirm the first point".	World frame
5	Jog the robot so that the TCP of the calibrated tool can point at the point on the X-axis of the desired work object frame, and then click "Confirm the second point".	The line between the second point and the first point is the X-axis of the work object frame.
6	Jog the robot so that the TCP of the calibrated tool can point at the point on the XY plane of the desired work object frame, and then click "Confirm the third point".	The user can also select a point on the desired Y-axis because the point on the Y-axis is also on the XY plane.

11.2.9 Tool frame

11.2.9.1 What is a tool?

Definition

A tool is a device mounted on the end-effector flange of the robot to complete a specific process. The common tools include **pneumatic/electric grippers**, **welding guns**, **sprinklers**, etc. No tool is attached to the robot when it is delivered from the factory, and you need to purchase or design appropriate tools according to the actual situation to complete the installation and setup in order to make the robot work.

Any tool should be calibrated before using it to get TCP (Tool center point) data.

When using external tools tools should be installed at the fixed position within the operating range of the robot instead of installing on the robot.

Explanation

A new tool needs to be defined before being used. In the xCore control system, the tool is saved and used through the data type of the tool. To define a tool means to create a tool-type variable. With regard to the details of the tool, please refer to the section of RL Programming Language (Tool). Simply speaking, we need to obtain the following tool-related parameters:

> ➢ TCP and orientation of tool (calibrating the tool frame);
> ➢ Mass, center of gravity, and rotational inertia (dynamics parameters of the tool);

The definition of the tool can only be modified through the HMI coordinates calibration interface. Please refer to the Calibration of tool coordinates for detailed steps. The tool-type variables can only be viewed but not created or modified in the variable management interface.

After the new frames are defined in the calibration interface, users can modify the tool's dynamic parameters using the manual input function or identify the tool's dynamic parameters through the parameter identification interface.

> **ℹ Notes**
>
> 1. Tool0 is a tool variable pre-defined by the system. Its tool coordinates coincide with the flange coordinates and both share the same dynamic parameter of 0.
> 2. The Tool0 variable is not allowed to be modified.

11.2.9.2 Tool center point

Definition

Tool Center Point (TCP) is a specific point on the tool which is normally used by a robot to carry out processing work, such as the wire tip of a welding gun, a tip of a pneumatic gripper, etc. The robot can rotate around the TCP and transform while keeping the position of the TCP unchanged.

Different tools may have different TCP, and determining appropriate TCP according to actual conditions can significantly increase programming efficiency.

TCP is also the origin of the tool frame. More details can be referred to from the introduction of tool variables.

<table>
<tr><td>ℹ</td><td>Notes</td></tr>
</table>

Unless otherwise specified, all references to "robot position, velocity, acceleration" in this Manual refer to the position, velocity, and acceleration of TCP relative to the work object frame.

Schematic diagram
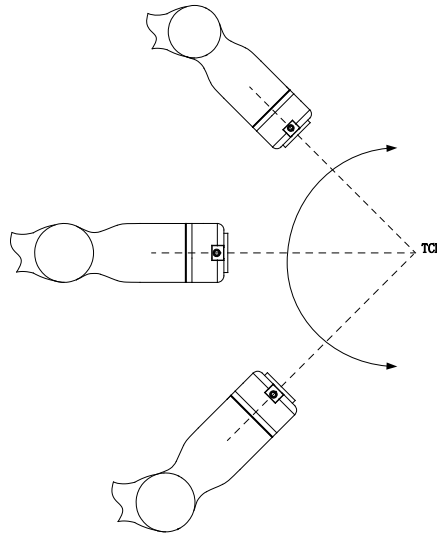
11.2.9.3 Tool frame

Explanation

The calibration of the tool frame refers to the process of measuring the position and orientation offsets of the tool frame relative to the flange frame.

If the manufacturer of the tool you are using provides these offset data, you can select "Manual Input" on the teach pendant and input directly without calibrating.

As to those tools without size data, the user needs to use the three methods offered by xCore to calibrate the tool frame.

➢ Four-point method, which is used to calibrate the origin of the tool frame;

➢ Three-point method, which is used to calibrate the orientation of the frame after calibration of the origin of the frame by the four-point method;

➢ The six-point method, which is used to calibrate the origin and orientation of the frame at the same time, is equivalent to the integration of the four-point method and the three-point method.

Calibration of the tool frame pose

Before the calibration of the tool frame, the user needs to prepare a fixed external point, which should be located within the robot's working range and can be contacted by the calibrated tool in a very flexible position and orientation. In the HMI tool calibration interface, there are detailed diagrams for reference.

	Operation	Description
1	Select a point on the tool to be calibrated and make this point as the origin of the tool frame, namely the Tool Coordinates Point (TCP).	Generally, TCP is always the processing point, for example, the wire tip of the arc welding gun, the fingertip of the claw, etc. It is also allowed to put the TCP on any part of the tool according to the actual situation.
2	Click "+" in the bottom right corner of the tool list to enter the New Tool Wizard interface.	Name the calibration tool.
3	Confirm whether the tool is a normal tool or an external tool.	Switch between normal tool/external tool according to different mounting method (external or handheld).
4	Select the six-point calibration method.	You can also select the four-point or three-point method. The four-point method calibrates only the tool origin while the three-point method calibrates only the tool frame orientation.
5	Select immediate load identification.	If the customer does not require tool load parameters, load identification can be omitted.
6	Jog the robot so that the selected TCP can be contacted by the external point and then click "Confirm the first point".	When two points are closing to each other, using the incremental mode can better adjust positions.
7	Repeat Step 6 until the four points are all confirmed.	To obtain higher calibration precision, the orientation difference between the four points shall be as high as possible, which means the robot should try to contact the external point in

		different orientations.
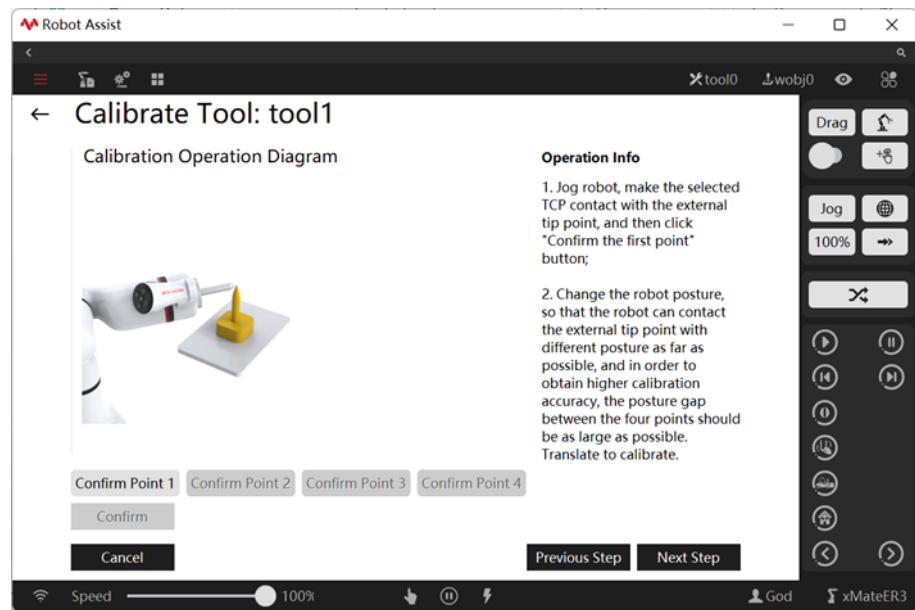
> ⚠ Warning
>
> If the robot is installed on the track, it is prohibited to operate the track during the calibration. Otherwise, the calibration will fail.

11.2.9.4 Tool load parameters

Explanation

As mentioned before, a complete definition of a tool needs to determine the kinematic parameter and dynamic parameter of the tool. The xCore system uses a load type variable to save the dynamic parameter of the object. As such the dynamic parameter of the tool is also called tool load. For details please refer to the introduction of variables tool and wobj. In particular, when an external tool is used, the corresponding work object load is saved in the load parameter in the tool variable.

Using the four-point method or six-point method can only determine the kinematic parameter of the tool. The dynamic parameter of the tool needs to be defined separately and there are two methods to define the load parameter of the tool:

➢ If there is data of tool load at hand, the user can select the manual input method on the tool frame calibration interface to input the corresponding data directly.

➢ If the load of the tool is unknown, the user can use the load identification function of the xCore system to identify.

Load identification

The load identification function can be conducive to the calculation of the dynamic parameters of the tool.

Steps for tool load identification:

➢ 1. Switch the robot to the Automatic mode and power on;

➢ 2. Run the load-free identification program with no load and wait for the program to complete;

➢ 3. Mount the tool load and run the load identification program, and wait for the program to complete;

> 4. When the identification is completed, the identification result pops up. Click to save.
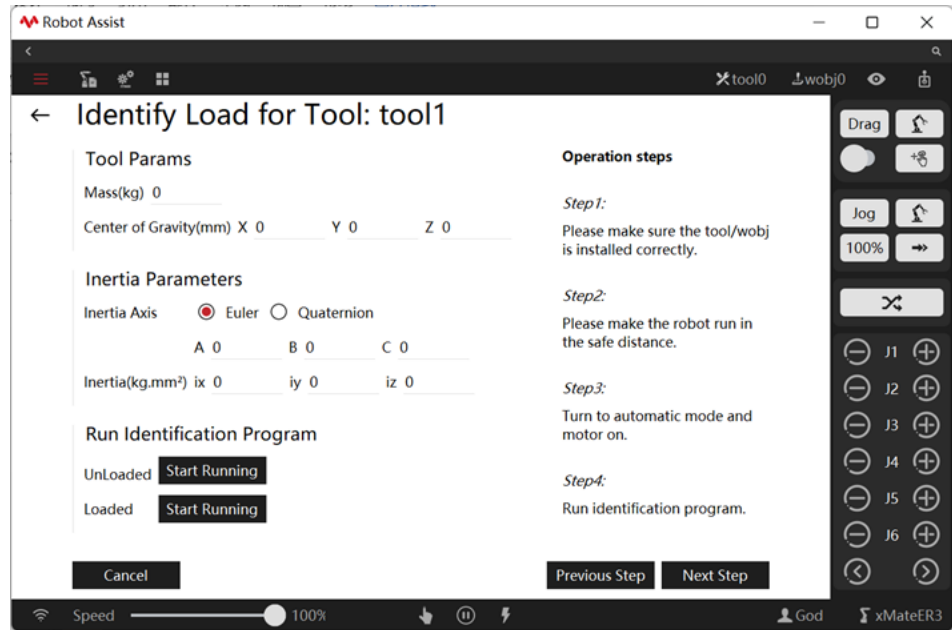
> **i** Notes

1. Please make sure to define the dynamic parameter of the new tool accurately. Otherwise, it will affect the motion of the robot and even damage the robot due to excessive load on some serious occasions.
2. Before the identification, switch and preheat the robot in advance for more than half an hour so as to raise the identification precision.
3. Load inertia calculation is based on the flange frame.
4. Load recognition is only supported in case of floor mounting.

Notes

The following circumstances during the identification will cause the identification to stop and all the received identification data lost. In this case, the user has to re-start the identification:

> User selects other tools or switches to other interfaces halfway through identification;
> User triggers the emergency stop or safety stop for external parts when the identification program is running;
> User switches from Automatic to Manual mode when the identification program is running.

> ⚠ Warning

The identification program needs to be executed under the Automatic mode, therefore all prevention measures should be effective. As the external control signal is able to start the robot at any time, please switch to the Automatic mode after the installation with personnel evacuated to a safe area.

11.2.9.5 Use of tools

Use when robot jogs

If it is necessary to use a special tool for jog operation, select the desired tool in the drop-down list of the 'Tool' in the menu on the upper side of the teach pendant interface.

Use in the program

> It is very simple to use a special tool in the program, simply use the desired tool in the 'Tool' parameter of the motion command. When programming the motion command in the 'Insert command' interface of the teach pendant, the 'Tool' and 'Work object' in default are consistent with those used during Jog operation, which means that the 'Tool' and 'Work object' in the menu at the upper side of the interface are currently selected. For the detailed operating steps, please refer to Insert command.

11.2.9.6 External tools

What is an external tool?

> Generally, we install tools on the robot and use them to complete the specified jobs using the robot motion. Such tools are called normal tools and include claw, suction cup, and welding gun.
> But in some special situations, installing a tool on the robot will affect the normal use, for example:
> 1.   The tool to be used is large or heavy, difficult to be installed on the robot, or probably affects the robot's motion.
> 2.   The work object to be processed is large and the working range of the robot cannot cover the whole work object in normal situations.
> 3.   Some special processing needs to be completed, for instance, grinding a square object needs the tool to revolve around 4 corners respectively.
>
> Under these circumstances, the effect of installing the work object on the robot while fixing the tool on a certain external place turns out to be better and more convenient. We call these tools that are installed outside the robot and fixed at a certain part the external tools (some brands call them Stationary Tool or Remote TCP).

Creation of external tool

> In the xCore system, the external tool is also described through the tool-type variables. There is a special mark robhold in the tool-type variables used to define if the tool is a normal tool or an external tool.
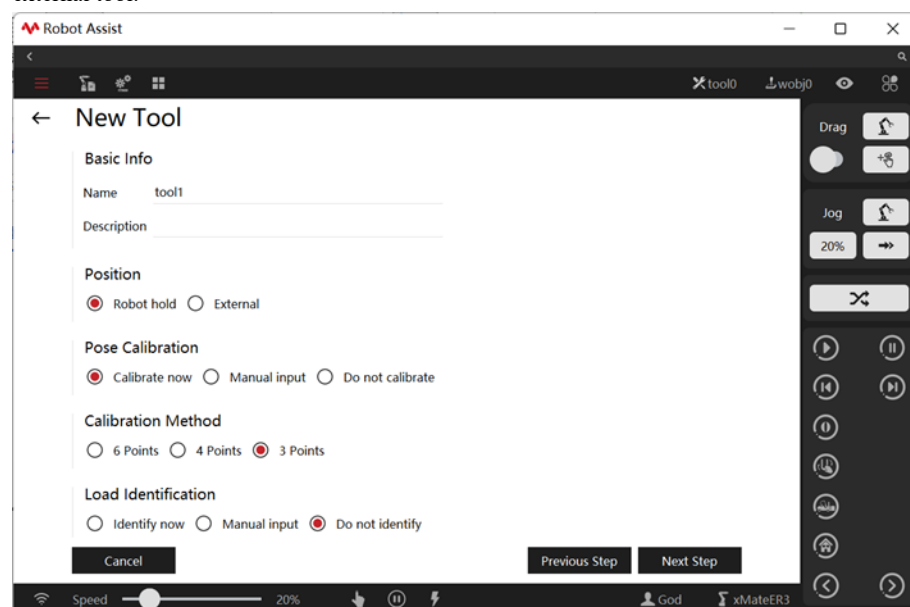
> It is very simple to use the teach pendant to create an external tool by selecting a certain tool in the tool calibration interface and then selecting External for Position.

Calibration of external tool frame

The external tool calibration is the same as the normal tool. It supports the four-point method, six-point method, and manual input. But calibrating the external tool frame needs the normal tool that has been already calibrated. We will take the four-point method here as an example.

	Operation	Description
1	Use admin to log in to the system and switch to the tool frame calibration interface.	
2	Calibrate a normal tool with a tip, or select a calibrated normal tool with the calibration precision as high as possible.	This normal tool is used for the later calibration of the external tool. This step can raise the precision of the external tool calibration effectively.
3	Jog the robot so that the TCP of the calibrated tool can point at the origin of the desired external tool frame and then click "Confirm the first point".	
4	Jog the robot so that the TCP of the calibrated tool can point at the origin of the desired work object in different orientations, and then confirm the second, third, and fourth points respectively.	The theory of how to select four points is the same as the four-point method for normal tool calibration.
5	After the calibration, the system will pop up the calibration error. Select whether to re-calibrate according to the error.	For information on calibration precision, please refer to Confirmation of calibration precision.

Notes

The external tool must be used together with the corresponding work object, meaning among the robhold parameters which are selected at the same time in the tool and work object respectively, one must be False while the other be True. Otherwise, the system will prompt an error and forbid jogging the robot.

When using the external tool, the reference for defining the tool frame and the work object frame is different from that for defining a normal tool. See the following form.

Frame	Definition of the normal tool relative to …	Definition of the external tool relative to …
Work object frame	User frame	User frame
User frame	World frame	Flange frame
Tool frame	Flange frame	World frame

For more details, please refer to the introduction of tool variables.

11.2.10 Work object frame list

11.2.10.1 What is a work object?

Explanation

Work object refers to the object that is being processed or handled by a robot with a tool.

The xCore system uses wobj (Work Object) type variables to describe an actual work object.

The introduction of the concept of work object is to simplify the programming steps and raise efficiency.

The motion trajectory of the robot is defined under this work object frame. There are two merits in doing so.

➢ When the work object moves or multiple identical work objects are being processed, the user only needs to recalibrate the work object frame instead of reprogramming since all the paths in the program will be updated accordingly;

➢ It allows the processing of the work objects that are moved by external axis (such as track, positioner, and so on).

Each work object actually contains two frames. One is the user frame relative to the work object, which can be considered as the bench/table where the work object is put. This is very useful in

processing multiple identical work objects. The other is the work object frame which is fixed on the work object. All program paths are described under the work object frame.
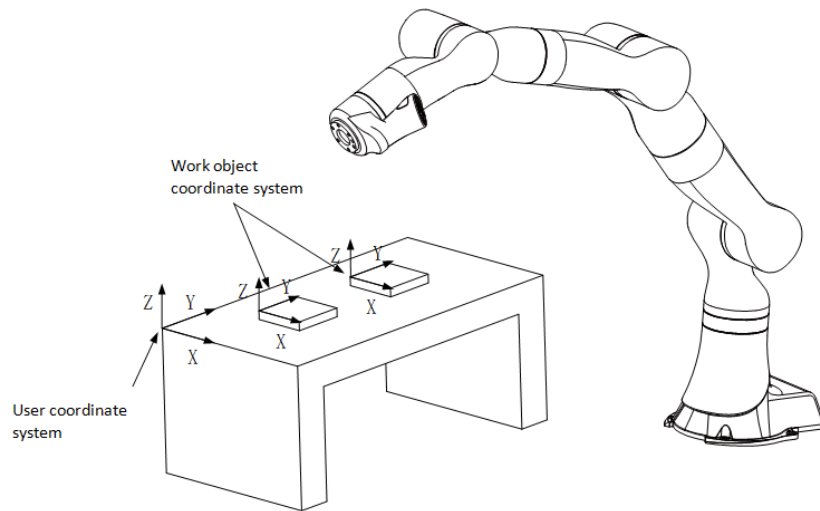
11.2.10.2 Definition of work object

Explanation

It is necessary to define a new work object before using it. In the xCore system, the work object is saved and used through wobj data. Defining a work object means creating a wobj variable.
The wobj variable does not contain any dynamic parameter, therefore the process of defining a work object is the process of calibrating the work object frame.

> **Notes**
>
> 1. Wobj0 is a work object variable pre-defined by the system. Its user coordinate and work object frame are all coincided with the world frame.
> 2. Same as tool0, wobj0 cannot be modified as well.
> 3. For PCB 3- or 4-axis robots, the work object frame only supports manual input. The components of orientation A and C are set to 0, and manual user modification is prohibited.

Calibration of work object frame

The method for calibrating the work object frame is the three-point method. Its operating steps are the same as the three-point method for calibration of the tool frame.
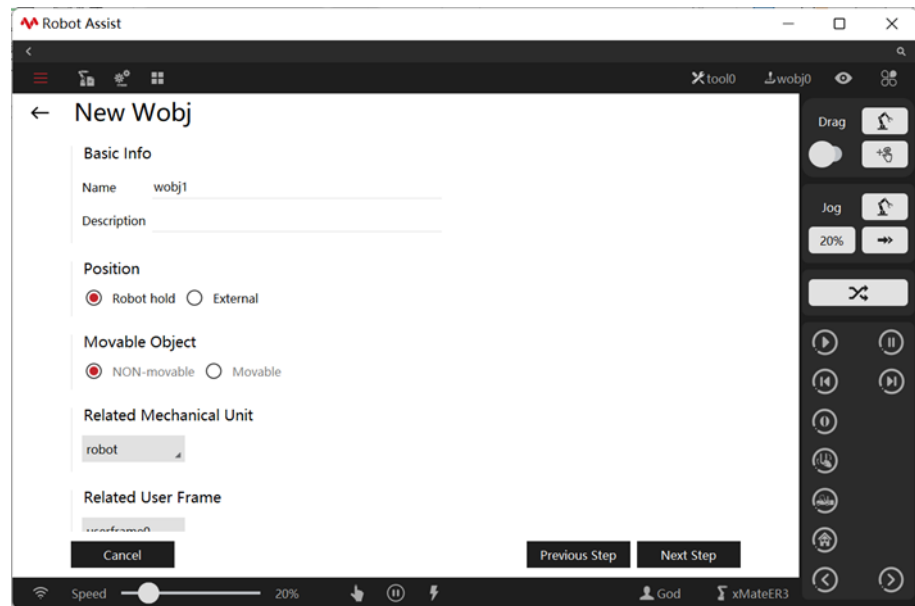
Before calibrating a work object, the user needs to calibrate a tool and then use the TCP of the tool to calibrate the work object frame.

For more convenient operation, it is recommended to use tools with tips.

	Operation	Description
1	Use admin to log in to the system and switch to the work object list interface.	
2	Name the work object frame to be calibrated in the name field.	The user frame is not mandatory. userframe0 is selected by default, i.e. the world frame.
3	Select external	See below for the calibration of the handheld work object
4	Jog the robot, make TCP of the calibrated tool point at the origin of the desired work object frame, and then click "Confirm the first point".	World frame
5	Jog the robot so that the TCP of the calibrated tool can point at the point on the X-axis of the desired work object frame, and then click "Confirm the second point".	The line between the second point and the first point is the X-axis of the work object frame.
6	Jog the robot so that the TCP of the calibrated tool can point at the point on the XY plane of the desired work object frame, and then click "Confirm the third point".	The user can also select a point on the desired Y-axis because the point on the Y-axis is also on the XY plane.

Calibration of handheld work object frame

For using the external tool function, the corresponding work object should be installed on the robot. In this case, this is called a handheld work object.

The handheld work object also needs the calibration of the work object frame and must use the calibrated external tool for calibration. For more details, please refer to the external tool function.

The general steps for calibration of handheld work object frame are as follows.

	Operation	Description
1	Use admin to log in to the system and switch to the work object frame calibration interface.	
2	Name the work object frame to be calibrated in the name field.	The user frame is not mandatory. userframe0 is selected by default, i.e. the world frame.
3	Select "Handheld"	
4	Jog the robot so that the TCP of the calibrated external tool can point at the origin of the desired	The line between the second point and the first point is the X-axis of the work

	work object frame and then click "Confirm the first point".	object frame.
5	Jog the robot so that the TCP of the calibrated external tool can point at the point on the X-axis of the desired work object frame, and then click "Confirm the second point".	The line between the second point and the first point is the X-axis of the work object frame.
6	Jog the robot so that the TCP of the calibrated external tool can point at the point on the XY plane of the desired work object frame, and then click "Confirm the third point".	The user can also select a point on the desired Y-axis because the point on the Y-axis is also on the XY plane.

11.2.10.3 Use of work object

Use when robot jogs

If it is necessary to perform Jog operation in a special work object frame, select the desired work object in the ⬐wobj0 drop-down list in the menu.

Use in the program

It is very simple to use a special work object in the program, simply use the desired work object in the "Work object" parameter. When programming the motion command in the 'Insert command' interface of the teach pendant, the 'Tool' and 'Work object' in default are consistent with those used during Jog operation, which means that the 'Tool' and 'Work object' in the menu at the upper side of the interface are currently selected. For the detailed operating steps, please refer to Insert command.

> **ℹ Notes**
>
> Generally, the work object parameter of the motion command is optional. As such, unless otherwise specified, the system will use wobj0 by default.
> The default wobj0 coincides with the world frame.
> To use the external tool function, all work object parameters corresponding to the tools must be designated.

11.2.10.4 Use of external tool/work object

Definition

To reduce the definition of default tools and work objects, whether the default tool tool0 and default work object wobj0 are handheld or not depends on the user-selected tool and work object:

1) When the user-selected tool frame, such as tool1, is handheld, the default work object frame wobj0 is automatically made external, and wobj0 coincides with the user frame; when the user-selected tool frame, such as tool1, is external, the default work object frame wobj0 is automatically made handheld, and wobj0 coincides with the flange frame;

2) The same goes for the work object frame. When the work object frame, such as wobj1, is external, the default tool frame tool0 is handheld and coincides with the flange frame; when the work object frame, such as wobj1, is handheld, the default tool frame tool0 is external and coincides with the user frame;

3) When both default tool frame tool0 and work object frame wobj0 are selected at the same time, tool0 is handheld and coincides with the flange frame, and wobj0 is external and coincides with the user frame.

3D interface display

Generally, users would like to display the pose of the manipulator end-effector in different frames, thus:

1) When using a handheld tool, the 3D interface shows the pose of the selected tool frame relative to the base/world/work object frames.

2) In the case of an external tool, the 3D interface shows the pose of the selected (handheld) work object frame relative to the base/world frames when the base/world frames are selected; and the pose of the selected (external) tool frame relative to the (handheld) work object frame when the work object frame is selected.
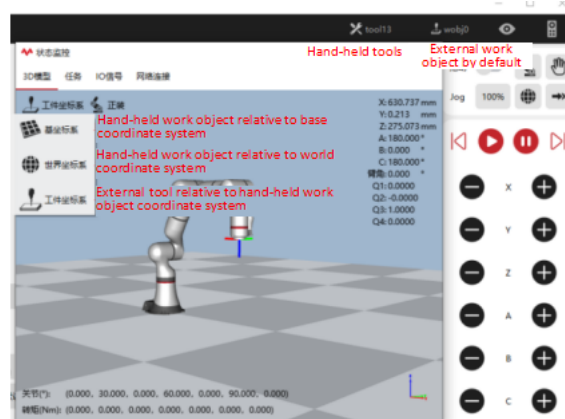
User frame

By definition, the user frame is also divided into two types: external and handheld, which depend on the corresponding work object frame. For example, when an external work object is used (corresponding tool frame is handheld), the user frame is automatically made external and represented in the world frame; when a work object handheld is used (corresponding tool frame is external), the user frame is automatically made handheld and represented in the flange frame.
When the user frame is used, the corresponding work object frame must be clearly distinguished. If the user frame is calibrated in the world frame, unexpected errors may appear when a handheld work object is used.

11.2.11 Vision System

Explanation

Vision task programming can be considered to be on the same level as RL motion task programming. After creating a new project, click xVision on the left side to open the vision task editing interface.
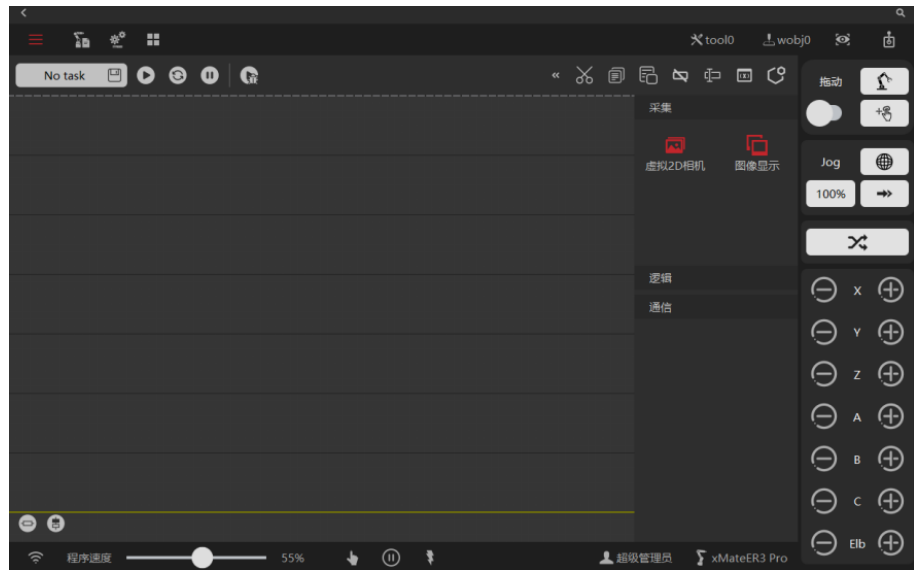
Create, open, and rename vision tasks

Vision task is also a kind of "task". Click "Task" to view, create, open, and rename visual tasks.
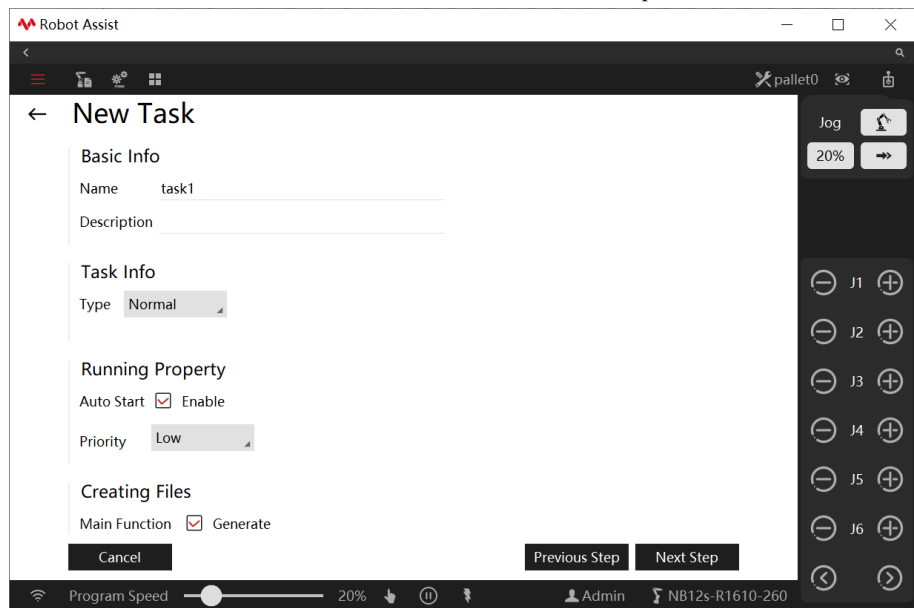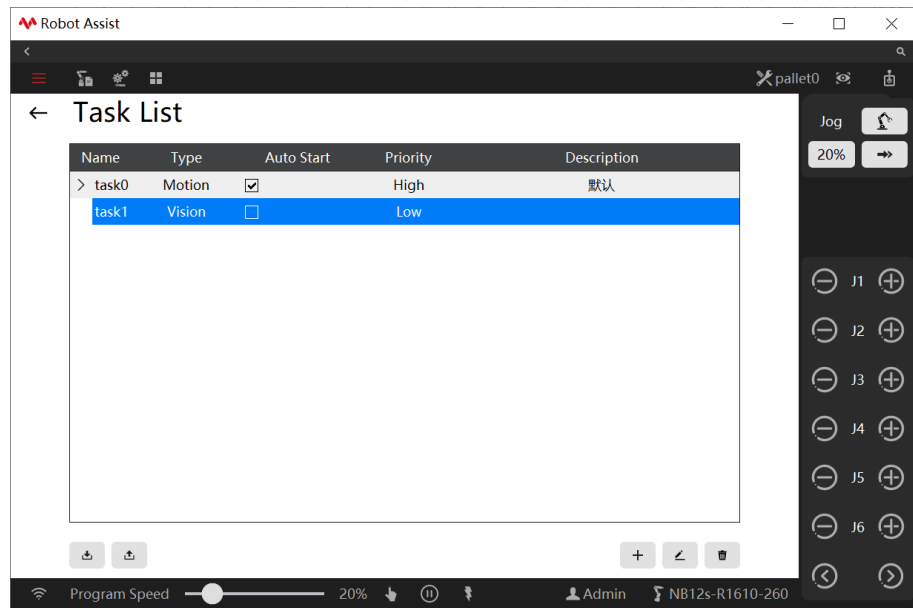
Visual tasks auto run on startup

Check the "Auto Start" attribute of the visual task, which is defined as the "auto run on startup" for the visual task.After the selected project is loaded, the vision task with the "Auto Start" property checked will be loaded and run in a loop automatically.

Note: Only one vision task can be checked.
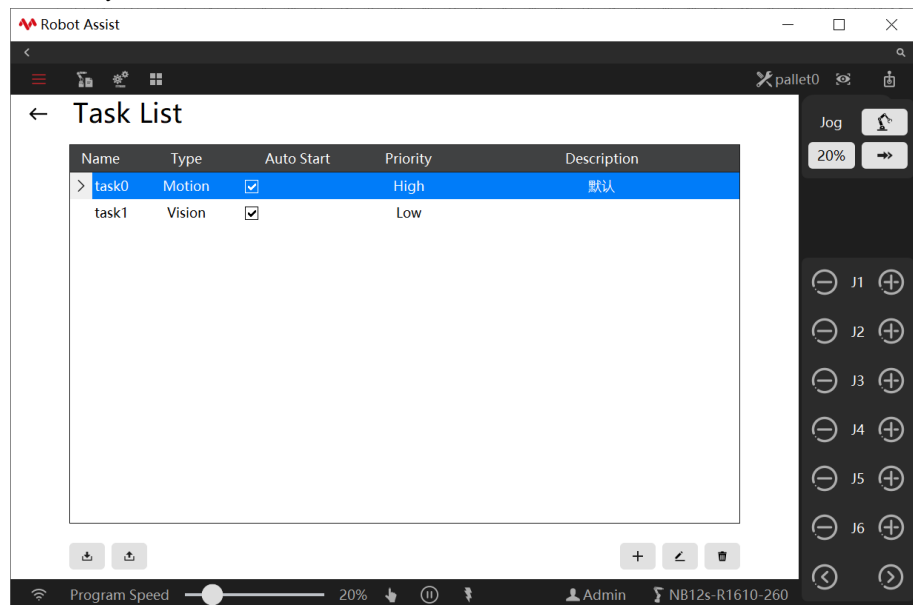
For more information on vision functions, please refer to *xVision User Manual*.

11.2 RL Programs

11.2.1 About RL language

Overview

Industrial robots are programmable devices that are suitable for many application scenarios. The language used to program robots is called Robot Language.

xCore system uses RL language as the programming language for all ROKAE robots.

RL Language is the abbreviation of ROKAE Robot Language. By using this language, users can program to control the robot through the teach pendant.

The RL language program file has a suffix of .mod, for example, MoveObj.mod or PickSomething.mod. Each program file forms a program module.

RL language commands are not case-sensitive. For example, MoveAbsJ, moveabsj and MOVEABSJ are all regarded as the correct MoveAbsJ command. However, in order to maintain a uniform language style, it is recommended to capitalize the initial letters.

Example

To demonstrate the features of the RL programming language, we can look at a simple program to understand the basic structure and format of RL:

Among them:

➢ The entire program is divided into two major sections, the declaration section, and the implementation section. The area before the first function in each Mod file is the declaration section. For example, in main.mod, the part before GLOBAL PROC main is the declaration section;

➢ VAR represents the storage type, indicating a variant. If the storage type is not declared, the RL program defaults it to a variable;

➢ int, robtarget, speed, zone, and tool are the special variable types of the RL language;

➢ MoveJ, MoveAbsj, and MoveL are standard motion commands in the RL language;

➢ Contents after "//" and "/**/" are comments.

11.2.2 Program structure

11.2.2.1 Overview

Explanation

All program files in the xCore system are grouped according to the concept of "Project". The following features are contained:

1. The RL program is divided into three levels according to the scope size:

a) Project, the highest level, configures the default robot parameters, manages sub-objects and tasks;

b) Task, contains several program modules;

c) Program modules, also known as the modules, are divided into program modules (.mod) and

system modules (.sys). A program file is a module;

d) Functions, also known as the ROUTINE, a program block for repeated calls that are defined by users;

2. A project can contain multiple tasks, each of which is independent and interacts with each other by the interfaces provided.

3. A program can contain multiple program modules, but there is only one main.mod, which contains a GLOBAL PROC main. The GLOBAL PROC main serves as the entry function of the entire project;

4. RL language support the function defined by users, which can either be saved in the same program file,or be saved in a different program file

5. The robot can only select one project for execution at a time.

The relationship among the project, program files, and functions are shown in the following figure:
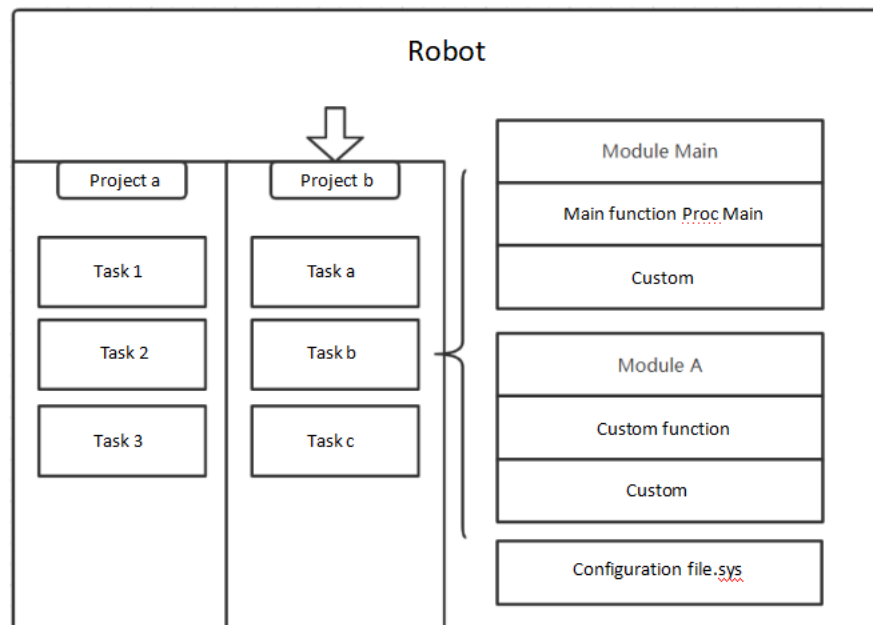
11.2.2.2 Program modules

Explanation	
	Program modules are either .mod or .sys files. Each program module contains a number of data variables and functions that are used to implement specific robotic functions. A project can contain multiple program files. Each program file can be copied and deleted, and other regular file operations are also allowed. In each project, there must be a program module that contains the main function that is used as the entry function for the entire project. Loading and executing a project is essentially executing the main function.
Module definition	
	The module is defined as: PROC main() … ENDPROC PROC test1() … ENDPROC PROC test2() …

ENDPROC

Notes

In each module, the code area located in front of the file and before the first function is called the declaration area. It is used to store variable declarations for the GLOBAL and LOCAL scopes. Users are not allowed to directly modify the area in the editor.

11.2.3 Program editing

11.2.3.1 Function menu

Explanation

To allow easy program debugging, the xCore system offers several powerful debugging features in the program editor interface.

Menu function
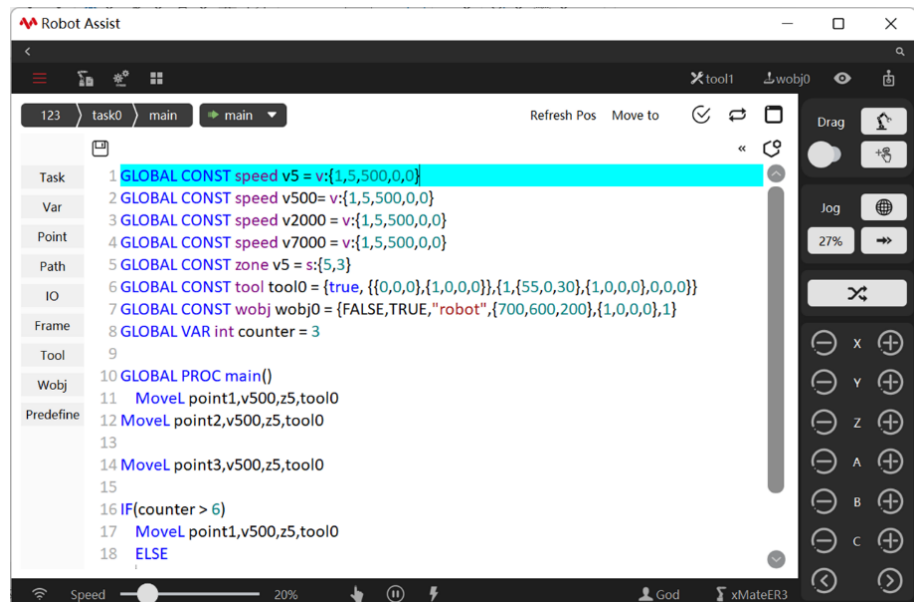
Program pointer to Main	Click [main ▼] to move the program pointer to the Main function, which is equivalent to program reset.
Program pointer to cursor	Click [光标 ▼] to move the program pointer to the line where the cursor is located.
Check program	Execute the program point to the Main check program to check whether there are certain obvious errors in the current program, such as the duplicate name of the function and missing key identifiers. It cannot check out all syntax errors.
Insert command	Click ↻ to insert motion commands and other commands.
Search program	Click ⌕ to search programs by keywords.
Comment command	Click {*} to comment on the selected code line. Multiple lines can be commented on at the same time.
Move down code line	Click ↓ to move the selected code line down one line. Multiple lines can be moved down at the same time.
Move code line up	Click ↑ to move the selected code line up one line. Multiple lines can be moved up at the same time.
Paste the entire line	Click ⧉ to insert the copied or cut content into the line of the cursor.

Copy the entire line	Click 🗐 to copy the selected line of code. Multiple lines can be copied at the same time.
Cut the entire line	Click ✂ to cut the selected line of code. Multiple lines can be cut at the same time.
Undo	Click ↶ to undo the previous operation.
Redo	Click ↷ to redo the previous operation undone.
Loop mode	Click ⇄ to select loop or run only once.
Output box	Click ▭ to display the printing information and syntax information.

11.2.4 Program debugging

11.2.4.1 Program pointer

Explanation

The program pointer points to the line that has been parsed and run by the program.
On the HMI interface, the program pointer is indicated by a small green arrow (also called the green pointer).

11.2.4.2 Motion pointer

Explanation

The motion pointer points to the current command the robot is executing;
On the HMI interface, the motion pointer is indicated by a red arrow.

11.2.4.3 Lookahead mechanism

Explanation

Lookahead means that the control system handles the subsequent program commands in advance when the robot is executing the current command during robot movement.
The introduction of the lookahead mechanism can be advantageous in the following aspects:
➤ Obtain the speed of the front trajectory, the acceleration requirements, and the constraints of the robot itself, so as to plan the optimal control strategy;
➤ Plan the turning trajectory of the turning zone according to the settings of the programmed turning zone;
➤ Acquire an abnormal state near the soft limit/boundary and singular points, etc., so that it can be processed in advance;

The lookahead mechanism cannot be turned off manually. The system automatically looks ahead when running the program. You can use the Program Pointer to view the lookahead position.
Some RL commands will interrupt the lookahead. When the interpreter encounters such a command, it will stop compiling until the robot executes the compilation of the corresponding command.
Only Print command, logical judgment command, and user-defined functions do not interrupt the lookahead mechanism, and all other functions will interrupt the lookahead mechanism.

11.2.4.4 Single-step debugging

Explanation

The single-step operation status is also known as Single-step Mode, as against the Continuous Mode. The robot can switch between the two modes in most cases.
Single-step running is mainly used for program debugging. The robot will execute commands of one line at a time and pause the program after commands are completed, which is convenient for

confirming whether teaching points meet requirements. When a multi-task project is being debugged, single-step debugging will only execute the tasks displayed on the HMI debugging interface, and the rest tasks will not be called.

If the single-step debugging executes read data commands (ReadDouble, ReadString, etc.), time-related commands (Wait, WaitUntil, etc.), and logic commands (IF, GOTO, etc.), it will take two to three clicks to complete the command due to the command characteristics.

Use restrictions

1. In Continuous mode where programs are executed automatically and the turning zone should be processed, motion lookahead is available.

2. In Single-step mode where commands are executed directly without processing the turn zone, motion lookahead is not available.

3. In Continuous Mode, motion only starts when there are enough lookahead points, and the system only continues to parse the command when the robot is in place.

4. In Single-step mode, all next-step signals are triggered by the interface, without turning zone processing and lookahead.

5. In Single-step Mode, no response is made when "Next" is clicked during motion.

6. In Continuous mode, callbacks during motion are responded to according to the lookahead logic.

7. The next step can go to any line and execute the command literally. RL programs only process "program commands", without distinguishing between motion commands and logic commands.

10. When the robot pauses on the turning zone in Continuous mode, the next step will go back to the target point corresponding to the current turning zone.

11.2.4.5 Regain path

Explanation

In some specific situations, the robot's position will deviate from its programmed path, for example:

➢ During the period when the program is stopped (excepting for program stop caused by program reset), the robot is moved to another position by Jog;

➢ The emergency stop is triggered when the program runs, and the robot executes STOP 0;

When the program starts again from the stop position, if the system detects that the robot has deviated from the programmed path,

the robot will then first perform a Regain Path motion to return to the original programming path.

To ensure safety, the movement speed of the robot is slower when returning to the programmed path, and the movement of the robot can be stopped at any time by pressing the "Stop" button on the teach pendant.

Use restrictions

The robot performs a joint trajectory when returning to the path, so the path of the end-effector is unpredictable. Please observe whether or not it collides with the surrounding environment.

Only when the robot continues to execute from stop point at the middle of the program, the control system will detect whether it deviates from the path.

If the deviation occurs, it will perform the regain path operation.

If the program is reset, then the system will not detect if it deviates from the path but will start executing directly from the first line.

Please be careful to prevent possible collisions.

11.2.4.6 Move program pointer

Explanation

If you need to start the program after a line from the middle of the program, you can use this function

to move the program pointer to the line where the cursor is, and then the program can be executed from a new position.

	Operation
1	Pause the running program, click the screen and move the cursor to the desired line.
2	In the program editor interface, click the "Debug" button and select "Program Pointer to Cursor".
3	The program pointer PP will be moved to the selected line.
4	After the program pointer PP points to the target line, click the program to start or go next. The robot then will slowly move from the current position to the target position of the specified line in the joint interpolation mode.

Use restrictions

There are the following restrictions for Move program pointer:

1. When using this function, the following commands will be ignored, and the compiler's compile position will be directly moved to the target line. In addition, all other commands will not be executed:

a) All motion commands;

b) SetDO, SetGO, Return, Wait, Print, and all Socket commands;

d) Function call line;

2. The condition of the flow control command is ignored when moving the program pointer.

3. Do not move the program pointer across functions. It is necessary to move the program pointer to the beginning of a function via "program pointer to function" first, and then use the pointer function of a program;

4. The pointer of a program can only be moved to the motion command line.

11.2.4.7 Variable management

Explanation

The variable management interface allows the creation, viewing, modification, and deletion of almost all variables in the robot system.

Currently supported variable types include: int/ byte/ bool/ double/ string/ robtarget/ jointtarget/ speed/ zone/ clock/ pose/ fcboxvol/intnum tasks.

Explanation

Although all types of variables can be entered by users manually in the programming interface, it is still recommended to use the dedicated interface for modification for the sake of convenient operation and fewer errors. You are advised to view in the variable management interface only.

The variable management interface is as follows:
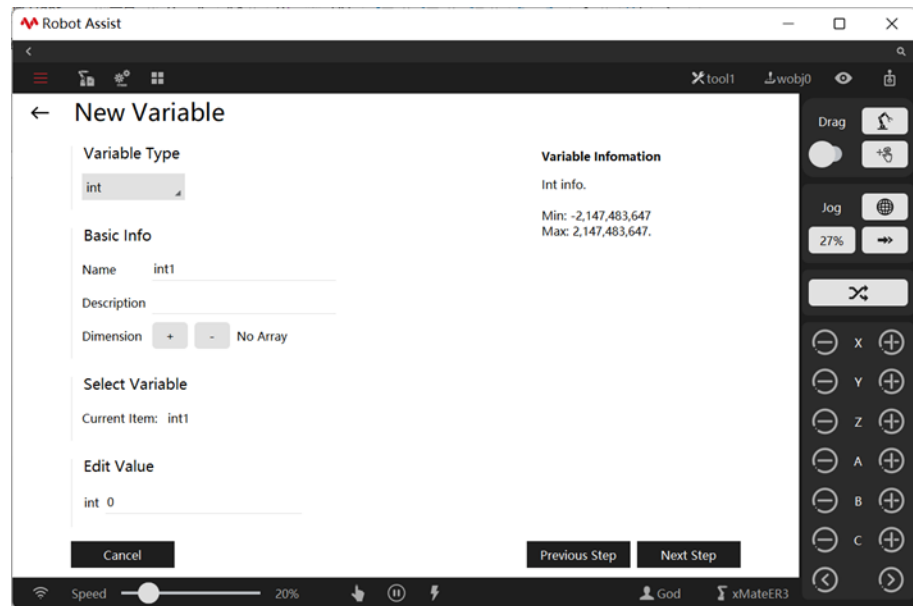
12 RL Programming Commands

12.1 Variables

12.1.1 Int

Explanation	
	The range of the integer int variable is $-2^{31} \sim 2^{31}$. It is recommended that the value is within the specified range. If the value is in excess of the range, it will be assigned randomly, and the maximum value range must not be exceeded when using it.
Example	
	For example, in the variable list, a variable is defined as follows:

It represents the data counter that defines an integer global variable type, and its initial value is 4.

12.1.2 uint

Explanation

The range of the integer uint variable is $0 \sim 2^{32}-1$. The maximum value range must not be exceeded when using it.

Example

Similar to a signed integer, in the variable list:

Variable Type

uint

Basic Info

Name   uint0

Description

Dimension   +   -   No Array

Select Variable

Current Item: uint0

12.1.3 Double

Explanation

Floating-point numbers are stored using 8 bytes. Do not exceed the value range when using them.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

double

Basic Info

Name   time

Description

Dimension   [+] [-]   No Array

Select Variable

Current Item: time

Edit Value

double 1.5

It represents the local variable time that defines a floating-point, and its initial value is 1.5.

12.1.4 Bool

Explanation

The variable bool is mainly used for status or logic judgments. The value is true or false.

When it is assigned an int or double value, non-zero takes the value of true and zero takes the value of false.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

bool

Basic Info

Name   ifClose

Description

Dimension   [+] [-]   No Array

Select Variable

Current Item: ifClose

Edit Value

bool true

It indicates that a bool type global variable ifClose is defined and the initial value is true.

12.1.5 String

Explanation

String-type variables consist of multiple letters or numbers and must be placed in double quotation marks "" at the time of defining.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

string

Basic Info

Name   name

Description

Dimension   +   -   No Array

Select Variable

Current Item: name

Edit Value

string "rokae"

It indicates that a string variable name is defined and initialized to "rokae".

String variables support the "+" operation for string concatenation.
Example: name = "Rok" + "ae"
It indicates that the variable name is assigned to "Rokae".

12.1.6 Array

Explanation

An array is a collection of variables with the same type, either one-dimensional or multi-dimensional. The elements in the array are accessed using subscripts. The subscript of each dimension begins with 1.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

int

Basic Info

Name   table

Description

Dimension   [+]   [-]   1   16

Select Variable

Current Item: table   [1]   [6]

Edit Value

int 8

It indicates that a two-dimensional array that contains 16 integer variables is defined. The value of the sixth element of line 1 is assigned to 8.

> **Notes**
>
> The total length of the array should not exceed 1000.

12.1.7 byte

Explanation

byte represents the unsigned byte in RL language, same as unsigned char in C++. The value range is 0~255, and negative values are not allowed. It is generally used in SocketSendByte command. When the byte value exceeds the limit, the lower 8 bytes will be truncated automatically without reporting an error.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

byte

Basic Info

Name   data

Description

Dimension   [+]   [-]   No Array

Select Variable

Current Item: data

Edit Value

byte 177

It defines a byte variable data, which has a value of 177.

> **i** Notes
>
> When the byte variable's value exceeds 255, it is automatically truncated, keeping only the lower 8 bits of the value, e.g. var byte data2=288, and the value of data2 is 32 after truncation.

12.1.8 clock

Explanation

The clock is used for timing, and clock-related commands are just like a stopwatch used for timing. The time accuracy of clock type storage is 0.001s, and the maximum time interval is 45 days (i.e., 45 x 24 x 3600 seconds).

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

clock

Basic Info

Name   clock0

Description

Dimension   +   -   No Array

Select Variable

Current Item: clock0

Edit Value

	The following example shows how to use variable clock:
Example 1	ClkStart clock1 ClkStop clock1 interval=ClkRead(clock1) ClkReset clock1 Interval (pre-declared double variable) reads the interval between ClkStart and ClkStop, in seconds (s).

12.1.9 Implicit type conversion

Explanation	Currently, during data setup in the variable lists, data types are restricted. Values that do not match the variable type cannot be successfully entered, thus avoiding type implicit conversion.
Example	For example, when defining the integer counter in the variable list, no decimals, only integers, can be entered.

12.1.10 confdata

Explanation	The confdata (Robot Configuration Data) is used to define the morphological configuration data that corresponds to the spatial target point. For a 7-axis robot with redundancy, the same Cartesian space target corresponds to a maximum of 8 different inverse kinematics when the elbow is the same, therefore, it is necessary to use confdata to specify the form to be selected. In addition, since the robot uses revolute joints, any one of the joints exhibits the same status at 1° and 361°. Therefore, after the form of the robot is selected, other methods are required to deal with the multiple-loop problem of the joint. Here, we use the quadrant method to mark the approximate range of joint angles. For example, when the joint angle is between 0 and 90 degrees, the quadrant number is 0. When the joint angle is between 90 to 180 degrees, it is marked as 1; by analogy, for every 90 degrees, the quadrant number is increased or decreased by 1. When the angle is negative, the corresponding number of quadrants is also negative, as shown in the following figure (left:

negative joint angle; right: positive joint angle). For robot joints, the angle increases when rotating anticlockwise and decreases when rotating clockwise. In the figure below, the joint angle decreases when a joint rotates clockwise, and the corresponding confdata changes as -1->-2->-3->-4 and 3->2->1->0, respectively.
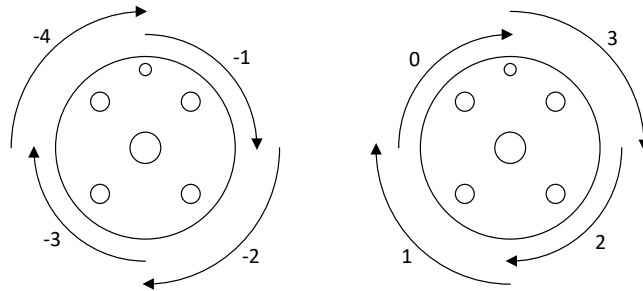
For xMate, 7 parameters are needed to complete the confdata, including:

➤ cf1, to record the number of quadrants of the Axis 1;

➤ cf2, to record the number of quadrants of the Axis 2;

➤ cf3, to record the number of quadrants of the Axis 3;

➤ cf4, to record the number of quadrants of the Axis 4;

➤ cf5, to record the number of quadrants of the Axis 5;

➤ cf6, to record the number of quadrants of the Axis 6;

➤ cf7, to record the number of quadrants of the Axis 7;

➤ cfx, to record which position the robot uses to reach the target position. See the explanation below for details.

Definition

cf1

    Data type: int

    The quadrant that corresponds to the Axis 1 angle.

Cf2

    Data type: int

    The quadrant that corresponds to the Axis 2 angle.

Cf3

    Data type: int

    The quadrant that corresponds to the Axis 3 angle.

Cf4

    Data type: int

    The quadrant that corresponds to the Axis 4 angle.

Cf5

    Data type: int

    The quadrant that corresponds to the Axis 5 angle.

Cf6

    Data type: int

    The quadrant that corresponds to the Axis 6 angle.

Cf7

    Data type: int

    The quadrant that corresponds to the Axis 7 angle.

cfx

    Data type: int

    The configuration number of the form used by the robot, ranging from 0 to 7.

Supplementary explanation

For xMate with redundant degrees of freedom, there are up to 8 different inverse kinematics for the same end-effector Cartesian space pose when the elbow remains the same. The values of cfx from 0 to 7 represent 8 groups of inverse kinematic solutions, which are explained in detail as follows.

cfx	Wrist center is on Axis 1...	Wrist center on the lower arm…	Axis 6 angle is...
0	Front	Front	Positive
1	Front	Front	Negative
2	Front	Rear	Positive
3	Front	Rear	Negative
4	Rear	Front	Positive
5	Rear	Front	Negative
6	Rear	Rear	Positive
7	Rear	Rear	Negative

12.1.11 jointtarget

Explanation

To store the robot's joint angle and the positions of external axes.

The unit of the joint angle is in degree, and the outer track is in mm.

Definition

robax

Angle of Robot Axis

Data type: double

robax contains 7 members of double type, which store the angle of the robot's 7 joints, in Degree.

extax

External Axis

Data type: double

The extax contains 6 members of double type and can store up to the position of 6 external axes.

If the external axis is a rotation axis, the unit is Degree; if the external axis is a linear axis, the unit is mm.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

jointtarget

Basic Info

Name   jointtarget0

Description

Dimension   [+]   [-]   No Array

Select Variable

Current Item: jointtarget0

Edit Value

robot_joint[0]   0

robot_joint[1]   0

robot_joint[2]   0

robot_joint[3]   0

robot_joint[4]   0

robot_joint[5]   90

ext_joint[0]   10

ext_joint[1]   0

ext_joint[2]   0

ext_joint[3]   0

ext_joint[4]   0

ext_joint[5]   0

The above command defines a point named "jointtarget0" in the joint space. Except that the Axis 5 is 90 degrees, the other axes of the robot are all 0 degrees. The first external axis is set to 10 degrees or 10 mm, depending on the type of external axis; the remaining external axes are set to zero.

12.1.12 load

Explanation

The variable type load is used to store the dynamic parameters of the robot load.

There are two main types of robot loads:

- ➢ The tool or work object itself installed at the end-effector of the robot;
- ➢ Objects that the tool picks up/sucks up.

The variable load does not support individual creation. It can only be manually modified in the tool calibration interface as a member of the tool-type variables or automatically modified by the control system using the load identification function.

By defining the dynamic parameters of the load correctly, the robot can achieve optimal performance.

⚠️   Warning

Be sure to correctly define the dynamic parameters of the end-effector load of the robot, including the tool itself and the two parts of the object captured by the tool. The wrong definition may lead to the following consequences:
- ➢ The robot cannot maximize the ability to use the servo system, resulting in degraded performance;
- ➢ The accuracy of the path is reduced, and the positioning error increases;
- ➢ Overloading of mechanical components results in a reduction in life or damage.

Definition

In the xCore system, the load is treated as a rigid body. There are four parameters for describing the load.

mass

Mass
Data type: double
It describes the mass of the load, in kg.

cogx

The offset of the center of mass in the X-direction.
Data type: double
If the tool is mounted on the robot, cogx records the offset of the center of mass in the X direction of the tool frame. If the external tool function is used, the cogx records the offset of the center of mass of the load held by the gripper in the X direction of the work object frame.

cogy

The offset of the center of mass in the Y direction.
Data type: double
If the tool is mounted on the robot, cogy records the offset of the center of gravity in the Y direction of the tool frame. If the external tool function is used, the cogy records the offset of the center of mass of the load held by the gripper in the Y direction of the work object frame.

cogz

The offset of the center of mass in the Z direction.
Data type: double
If the tool is mounted on the robot, cogz records the offset of the center of gravity in the Z direction of the tool frame. If the external tool function is used, the cogz records the offset of the center of mass of the load held by the gripper in the Z direction of the work object frame.

q1~q4

Quaternion, to record the direction of the principal axis of inertia of the load.
Data type: double
When the tool is mounted on the robot, the orientation of the principal axis of inertia is described in the tool frame. See the figure below for details:
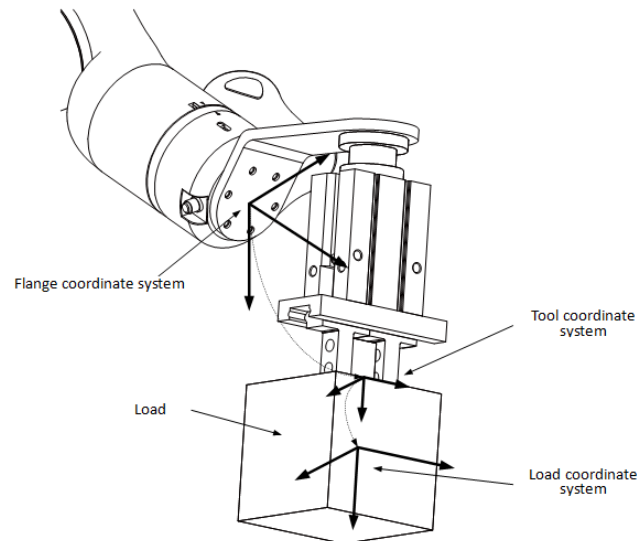
When using an external tool, the direction of the principal axis of inertia is described in the work object frame. See the figure below:
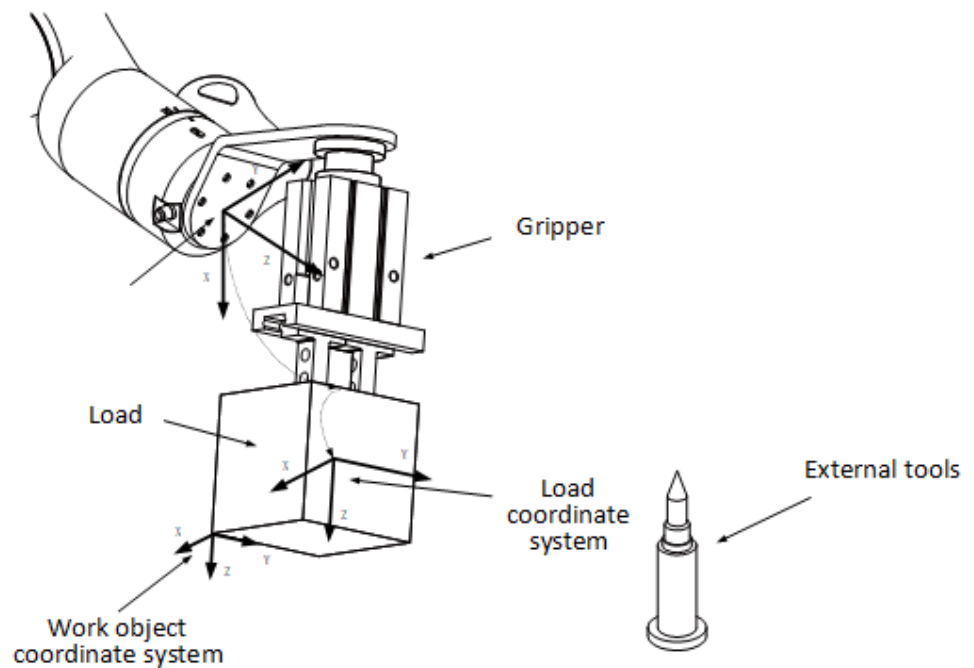
ix

Inertia x

Data type: double

The inertia of the load along the x-axis, in kgm^2.

Correctly defining the load inertia helps to improve the robot's movement accuracy, especially when handling large objects. If ix, iy, iz are set to zero, the load will be treated as a mass point.

Usually, if the distance from the center of mass of the load to the flange center point is smaller than the maximum size of the load itself, the load inertia should be defined, as shown in the following figure:
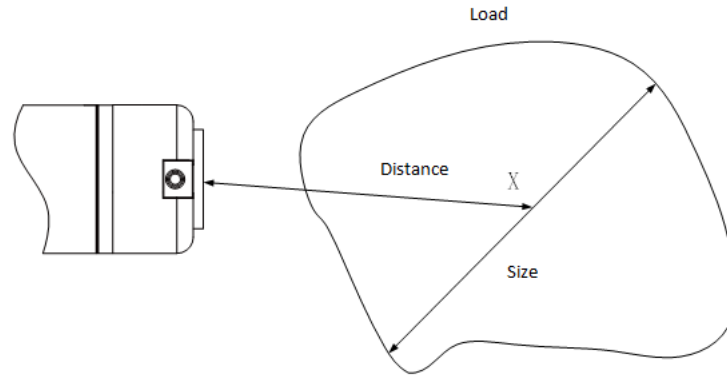
iy

Inertia y

Data type: double

The inertia of the load along the y-axis, in kgm^2.

iz

Inertia z

Data type: double

The inertia of the load along the z-axis, in kgm^2.

12.1.13 orient

Explanation

To store the orientation information of the frame or space rigid body.

Variables of type orient do not support individual creation or modification and are only used as member variables of some variables.

Definition

The RL language system uses quaternions to represent orientations, so there are a total of 4 components expressed as follows:

q1

Data type: double

The 1st component of the quaternion.

q2

Data type: double

The 2nd component of the quaternion.

q3

Data type: double

The 3rd component of the quaternion.

q4

Data type: double

The 4th component of the quaternion.

About the quaternions

We usually describe the orientation of the rigid body by using the rotation matrix. The quaternion is another way to describe orientation more concisely.

The four components of the quaternion satisfy the following relationship:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

The rotation matrix and the quaternion can be converted to one another. It is supposed that there is a rotation matrix R,

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

then:

$q_1 = \dfrac{\sqrt{r_{11} + r_{22} + r_{33} + 1}}{2}$	\
$q_2 = \dfrac{\sqrt{r_{11} - r_{22} - r_{33} + 1}}{2}$	$sign\ q_2 = sign(r_{32} - r_{23})$
$q_3 = \dfrac{\sqrt{r_{22} - r_{11} - r_{33} + 1}}{2}$	$sign\ q_3 = sign(r_{13} - r_{31})$
$q_4 = \dfrac{\sqrt{r_{33} - r_{11} - r_{22} + 1}}{2}$	$sign\ q_4 = sign(r_{21} - r_{12})$

12.1.14 pos

Explanation

It is used to store location information in 3D space.

Variables of pos type do not support individual creation or modification and are only used as member variables of some variables.

Definition

As the RL language system describes three-dimensional space using the Cartesian frame, so the pos variable has three components: x, y, and z.

x

Data type: double

The X coordinate of the position.

y

Data type: double

The Y coordinate of the position.

z

Data type: double

The Z coordinate of the position.

12.1.15 pose

Explanation

To store the position and orientation of Cartesian space.

Definition

X

Data type: double

The X coordinate of the position.

Y

Data type: double

The Y coordinate of the position.

Z

Data type: double

The Z coordinate of the position.

Q1

Data type: double

The 1st component of the quaternion.

Q2

Data type: double

The 2nd component of the quaternion.

Q3

Data type: double

The 3rd component of the quaternion.

Q4

Data type: double

The 4th component of the quaternion.

12.1.16 robtarget

Explanation

Cartesian positions and orientations for storing 3D space, which is used for MoveJ, MoveL, MoveC, and MoveT commands.

Because of the multi-solvability of the inverse kinematics of the robot, the robot can arrive in many different forms for the same target pose. In order to clearly specify the configuration form, the robtarget variable also contains the robot configuration data.

Variables of the robtarget type are automatically created when the motion command is inserted by auxiliary programming. Manually changing the internal value of the variable may lead to the non-correspondence between the Pose and ConfData, and the robot cannot execute the motion command normally.

> ⚠ Warning
>
> The use of Cartesian positions and orientations in robot programs is defined in the **work object frame**. If the work object used in the end is not the same as that used during the initial programming, the robot's motion will deviate from the desired path. Therefore, it shall be confirmed that the changes in work object will not cause danger in the following two cases:
> - Use the "Modify Command" function to change the wobj parameter of the command;
> - The actual work object used is different from the one used in the program commands.
> Improper use can result in personal injury or equipment damage!

Definition

trans

Spatial position

Data type: pos

The position offset stored in the reference frame.

rot

Orientation

Data type: orient

The orientation stored in the reference frame.

conf

Robot Configuration

Data type: confdata

To save the configuration data of the robot. Please refer to confdata for details.

extax

External Axes

Data type: double

The extax contains 6 members of double type and can store up to the position of 6 external axes.

If the external axis is a rotation axis, the unit is Degree; if the external axis is a linear axis, the unit is mm.

Example

For example, in the variable list, a variable is defined as follows:

New Variable

Variable Type

robtarget

Basic Info

Name   robtarget0

Description

Dimension   [+]   [-]   No Array

Select Variable

Current Item: robtarget0

Edit Value

X 0

Y 0

Z 0

Q1 0

Q2 0

Q3 0

Q4 0

cf1 0

cf2 0

cf3 0

cf4 0

cf5 0

cf6 0

cf7 0

cfx 1

ext joint[0] 0

ext joint[1] 0

ext joint[2] 0

ext joint[3] 0

ext joint[4] 0

ext joint[5] 0

A Cartesian space pose named p1 with the position and orientation (in quaternions) as shown above is defined. The elbow is 10°, and the angles of the Axis 1, 3, 5, and 7 are between 0 and 90°. The robot belongs to the first group of morphological configurations (see confdata for details), and all external axes are in zero.

12.1.17 signalxx

Explanation	
	Signalxx type variables are used to describe I/O signals. All signalxx type variables need to be defined in the "Input/Output" and then used in the program. Direct declaration in the program is not supported.

Description	
	Signalxx currently only supports digital input and output, including the following variable types:

Variable type	Used to describe...	Description
signaldi	Digital input signal	The value is True or False, and only indicates the status
signaldo	Digital output signal	The value is True or False and is assigned to output

signalgi	Digit group input signal	A segment of continuous physical input port is defined as a binary number that can be converted to decimal for use in RL. It supports up to 16 DIs to constitute the group input. Therefore, the value of signalgi ranges from 0 to (2^n -1), with n as the number of DI points contained in group input
signalgo	Digit group output signal	A segment of continuous physical output port is defined as a binary number that can be converted to decimal for use in RL. It supports up to 16 DOs to constitute the group output. Therefore, the value of signalgo ranges from 0 to (2^n -1), with n as the number of DO points contained in the group output

The signaldo and signalgo types contain only signal references and can be assigned using separate commands (e.g. SetDO, SetGO, etc.).

Signaldi and signalgi can be used to directly obtain the value of the corresponding input signal in the program.

Example

Example 1

//Use the state of the digital input as a criterion for judgment
IF (di1 == true)
    do something…
ENDIF

Example 2

//Use the state of the digital group input as a criterion for judgment
For example, if the definition group input gi2 maps the first three bits of the 1st byte of Profinet IO, then when the values of bit0 to bit2 are 0, 1, and 1, the value of gi2 is 110 (6 after being converted to int). The same goes for group output (signalgo) as well.
IF (gi2 == 8)
    do something…
endif

Notes

➤ It is not supported to define/declare variables of type signalxx in the program. If such usage occurs, the program will report an error. Before using variables of signalxx type, please configure them in the IO signal list.

Notes

1. The scope of the signalxx variable is System, and its priority, when compared with other scope types, is System> GLOBAL> LOCAL.
2. If the variables declared in the Signal of the IO configuration interface and in the RL programs have the same name, the variable of scope in a lower level will be selected.

12.1.18 speed

Explanation

To define the speed of the robot and the external axes.

For users' convenience, the system presets the commonly used speed variables, which can be directly selected through auxiliary programming. For details, please refer to Insert Command.

Definition

The speed-type variable contains 5 member variables: Joint Velocity Percentage, TCP Linear Velocity, Orientation Velocity, External Axis Linear Velocity, and External Axis Angular Velocity.

Joint Velocity Percentage

Data type: double
It is used to specify the motion speed when the joint movement command is applied. It is applicable to the commands MoveAbsJ and MoveJ. The value ranges from 1% to 100%.

TCP Linear Velocity

> Data type: double
>
> It is used to define the linear velocity of the TCP. The value ranges from 0.001 mm/s to 7000 mm/s.

Orientation Velocity

> Data type: double
>
> It is used to define the rotation speed of the tool, ranging from 0.001 degrees/s to 500 degrees/s.

External Axis Linear Velocity

> Data type: double
>
> It is used to define the motion speed of the external linear axis, ranging from 0 mm/s to 5000 mm/s.

External Axis Angular Velocity

> Data type: double
>
> It is used to define the motion speed of the external rotary axes, ranging from 0 degrees/s to 1000 degrees/s.

Example

> In the variable list:

> The image above shows a definition of a speed variable named speed0, in which the joint rotation speed is 40% of the maximum allowable speed, the TCP linear speed is 300 mm/s, the space rotation speed is 100°/s, and the external axis angular velocity is 200°/s, and the external axis linear velocity is 1,000 mm/s.

Predefined speed variables

> The system predefines some common speed variables, as shown in the following table.

Name	Joint Velocity Percentage	TCP Linear Velocity	Orientation Velocity	External Axis Angular Velocity	External Axis Linear Velocity
v5	1%	5 mm/s	200°/s	1000°/s	5000 mm/s
v10	3%	10 mm/s	200°/s	1000°/s	5000 mm/s
v25	5%	25 mm/s	200°/s	1000°/s	5000 mm/s
v30	5%	30 mm/s	200°/s	1000°/s	5000 mm/s
v40	5%	40 mm/s	200°/s	1000°/s	5000 mm/s
v50	8%	50 mm/s	200°/s	1000°/s	5000 mm/s
v60	8%	60 mm/s	200°/s	1000°/s	5000 mm/s
v80	8%	80 mm/s	200°/s	1000°/s	5000 mm/s
v100	10%	100 mm/s	200°/s	1000°/s	5000 mm/s
v150	15%	150 mm/s	200°/s	1000°/s	5000 mm/s
v200	20%	200 mm/s	200°/s	1000°/s	5000 mm/s
v300	30%	300 mm/s	200°/s	1000°/s	5000 mm/s
v400	40%	400 mm/s	200°/s	1000°/s	5000 mm/s
v500	50%	500 mm/s	200°/s	1000°/s	5000 mm/s
v600	60%	600 mm/s	200°/s	1000°/s	5000 mm/s
v800	70%	800 mm/s	200°/s	1000°/s	5000 mm/s
v1000	100%	1000 mm/s	200°/s	1000°/s	5000 mm/s
v1500	100%	1500 mm/s	200°/s	1000°/s	5000 mm/s
v2000	100%	2000 mm/s	200°/s	1000°/s	5000 mm/s
V3000	100%	3000 mm/s	200°/s	1000°/s	5000 mm/s
v4000	100%	4000 mm/s	200°/s	1000°/s	5000 mm/s
v5000	100%	5000 mm/s	200°/s	1000°/s	5000 mm/s
v6000	100%	6000 mm/s	200°/s	1000°/s	5000 mm/s
v7000	100%	7000 mm/s	200°/s	1000°/s	5000 mm/s
vmax	100%	infinite	200°/s	1000°/s	5000 mm/s

> **Notes**
>
> All space rotation speeds in the system's pre-defined speed variable are 200°/s. If there are special requirements on the rotation speed of the end-effector of the robot, a new speed variable can be defined for use according to the process requirements.

12.1.19 tool

Explanation

The tool-type variables are used to record tool parameters, including TCP, orientation, and dynamic parameters of the tools used by the robot.

The robot uses tools to interact with the outside world, so the tool variable will affect the motion of the robot from the following aspects:

➢ Only the TCP will move according to the programmed path and speed. When the robot executes a pure spatial rotation, only TCP will remain motionless;

➢ The motion path and speed specified during programming refer to the path and speed of the tool frame relative to the work object frame. Therefore, replacing a well-calibrated tool or work object does not affect the shape and speed of the path;

➢ When using external tools, the speed of programming refers to the speed of a work object (relative to external tools).

Note that when using the external tool, tframe in the tool-type variable will record the zero position and orientation offset of the external tool, while tload will record the dynamic parameters of the gripper that is installed at the end-effector of the robot for grasping work object.

The data of the tool-type variable is stored in the database. When the program is loaded, it is read by the program editor from the database. Therefore, do not try to modify the tool-type variable directly in the program editor, and thus the unpredictable errors will be avoided. If you need to modify the tool-type variable, please modify it through the calibration interface. See the Calibration of the tool frame for details.

> **Warning**
>
> Be sure to correctly define the dynamic parameters of the end-effector load of the robot, including the tool itself and the two parts of the object captured by the tool. The wrong definition may lead

> to the following consequences:
> ➢ The robot cannot maximize the ability to use the servo system, resulting in degraded performance;
> ➢ The accuracy of the path is reduced, and the positioning error increases;
> ➢ Overloading of mechanical components results in a reduction in life or damage.

Definition

robhold

Data type: bool

It is used to define whether the tool is installed on the robot. True indicates that the tool is installed on the robot. False indicates that the tool is not installed on the robot and an external tool is being used.

When making a jog or executing a program, only one of the robhold parameters can be True in the tool/work object combination used at the same time. That is, if the robhold of the tool is True, the corresponding work object robhold must be false, and vice versa, otherwise, the robot will prompt an error, and it is impossible to make a jog or execute the corresponding program command.

tframe

Tool frame

Data type: pose

Record the tool frame of the tool used, including:

➢ TCP represents the offset in the x, y, and z directions relative to the robot end-effector flange frame, in millimeters.

➢ The orientation offset of the tool frame relative to the flange frame is expressed in quaternion. See the following figure for details:
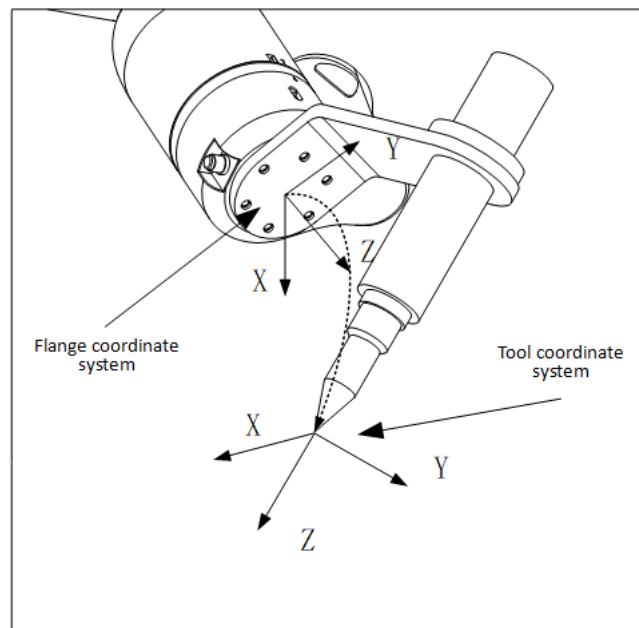
> **i** Notes
>
> When using the external tool function, the TCP and orientation are defined relative to the world frame.
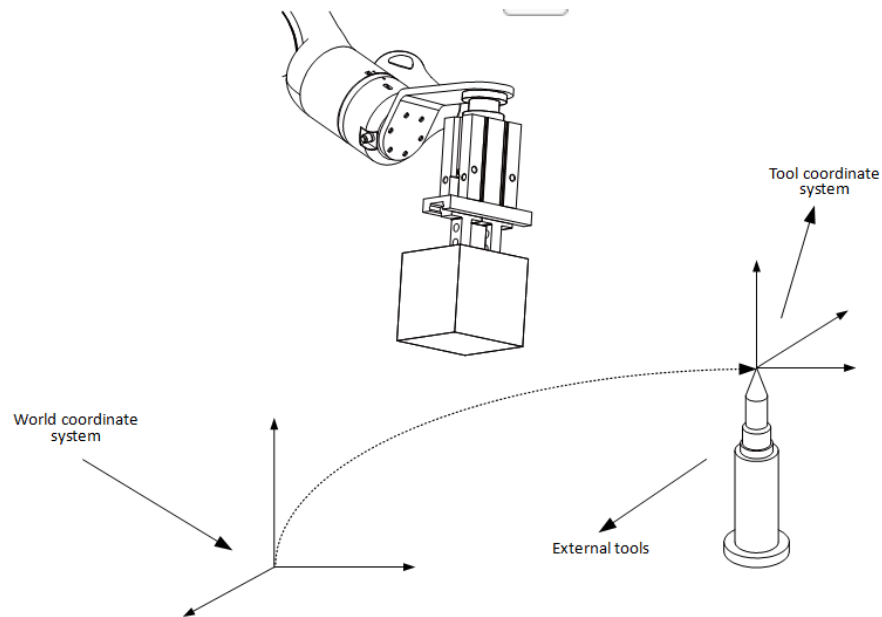
tload

Dynamic parameters of the tool

Data type: load

To record the dynamic parameters of the tool. For the common tool, tload describes the dynamic parameters of the entire tool. For external tools, tload describes the dynamic parameters of the gripper used by the robot (holding the work object).

For general tools installed on the robot, the load parameters include:

➢ The mass of the tool (weight), in kg;
➢ The center of gravity of the tool, described in the flange frame, in millimeters (mm);
➢ The direction of the principal axis of inertia, described in the flange frame; and
➢ The inertia magnitude of the tool along the principal axis of inertia, in kgm^2. If all inertia components are defined as 0 kgm^2, the tool is treated as a Point Mass.

> **ℹ Notes**
>
> If the robot is using an external tool, then the tload member is used to record the dynamic parameters of the gripper installed on the robot. The meaning of the specific parameters remains unchanged.

> **ℹ Notes**
>
> Please note that the tload members only define the dynamic parameters of the gripper used by the robot (holding the work object). The dynamic parameters of the gripped work object are not included. To ensure that the robot performs optimally under all circumstances, you need to define two tool variables to handle this situation:
> ➢ A tool saves all parameters of the gripper itself;
> ➢ Another tool saves all parameters of the gripper + gripped work object;
> The use of different tools in the motion command would help implement the switching function with or without load.

Example
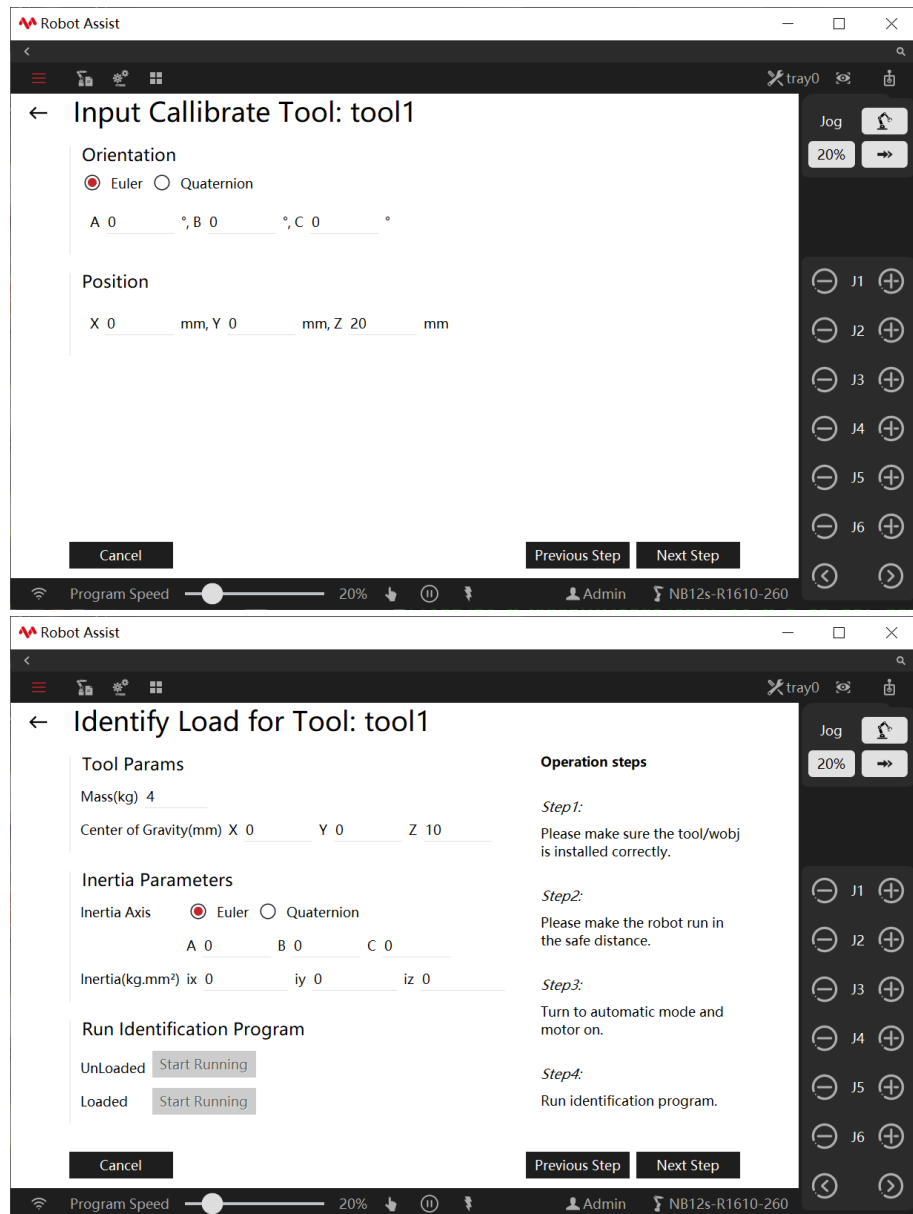
A work object named tool2 is defined, where the parameters are:

➢ The tool is mounted on the robot;

➢ TCP offsets in the XYZ directions relative to the flange frame are 100, 0, 220, and the orientation is the same as the flange frame.

➢ The mass of the tool is 2kg, and the offset of the center of mass relative to the origin of the flange frame in the XYZ directions are 20, 0, 50mm, respectively;

The tool is treated as a mass point and the inertia data is zero.

12.1.20 trigdata

Explanation

trigdata is used to store information data about the trigger events during robot motion, including trigger conditions and trigger actions.

The trigger condition is usually reaching a specified location on the path; the trigger action can be setting IO, setting variables, etc.

Variables of type trigdata cannot be defined by the assignment operator and can only be defined by a specific RL command, so the information stored in each trigdata variable depends on the Trig command as used, for example, the TrigIO, etc.

Then, it can be used by the corresponding movement commands TrigL, TrigC, TrigJ, etc.

Example

Example 1

The following example shows how to use the trigdata:

VAR trigdata gripopen
TrigIO gripopen,0.5,do1,true
TrigL p1,v500,gripopen,fine,tool1

12.1.21 wobj

Explanation

wobj is an abbreviation for Work Object. Work object refers to an object processed, handled, or transported by a robot.

All the positions used in the motion command are defined in the work object frame (if no work object frame is specified, it defaults to the world frame. The world frame can be seen as a wobj0). There are several benefits in doing this:

➢ The location of many processing points can be obtained from the design drawing of the work object and used directly;

➢ When the robot is reinstalled or the work object is moved, you only need to re-calibrate the work object frame to reuse the previous program and avoid reprogramming.

➢ With a suitable sensor provided, vibrations or slight movements of the work object can be automatically compensated.

Under normal circumstances, if you do not define a specific work object frame, the control system will then regard the world frame as the default work object frame wobj0. However, when using external tools, the work object frame must be defined because the programming path and speed refer to the path and speed of the work object, rather than the tool.

Usually, the work object frame is defined relative to the user frame, but if the user does not specify a user frame, the work object frame is defined by default relative to the world frame. For details, see the Robot's frames.

The work object actually consists of two frames, the user frame and the work object frame. Inserting a user frame at the upper layer of the work object frame is to support the situation where multiple identical work objects need to be machined. For an explanation of the defining relationships of the relevant coordinates, see the explanation of oframe in the "Definitions" section.

> **ℹ Notes**
>
> The data of the wobj-type variable is stored in the database. When the program is loaded, it is read by the program editor from the database. Therefore, do not try to modify the wobj-type variable directly in the program editor, and thus the unpredictable errors will be avoided. If you need to modify the wobj-type variable, please modify it through the Calibration interface. For details, please refer to the Definition of the work object.

Definition

robhold

It is used to define whether the work object is mounted on the robot. True indicates that the work object is mounted on the robot and the external tool is currently being used. False indicates that the work object is not mounted on the robot and the normal tool is currently being used.

ufprog

User Frame Programmed
Variable type: bool
It is used to define whether the user frame is fixed or moving. True indicates that the user frame is fixed, False indicates that the user frame is moving, e.g., defines whether it is on an external positioner or another robot.

This value is mostly used when the robot is required to coordinate its movement with the positioner or other robots.

ufmec

User Frame Mechanical Unit
Data type: string
The mechanical unit name is used to specify which mechanical unit the user frame is bound to. It is useful only if ufprog is false.

oframe

Work Object Frame
Data type: pose
It is used to store the origin and orientation of the work object frame.

uframe_id

Id of User Frame
Data type: int
It is used to store the id of the user frame. The corresponding user frame can be found by id.
When using **normal tools** (non-external tools), the frame definition chain is as follows:
➢ The work object frame is defined relative to the user frame;
➢ The user frame is defined relative to the world frame.
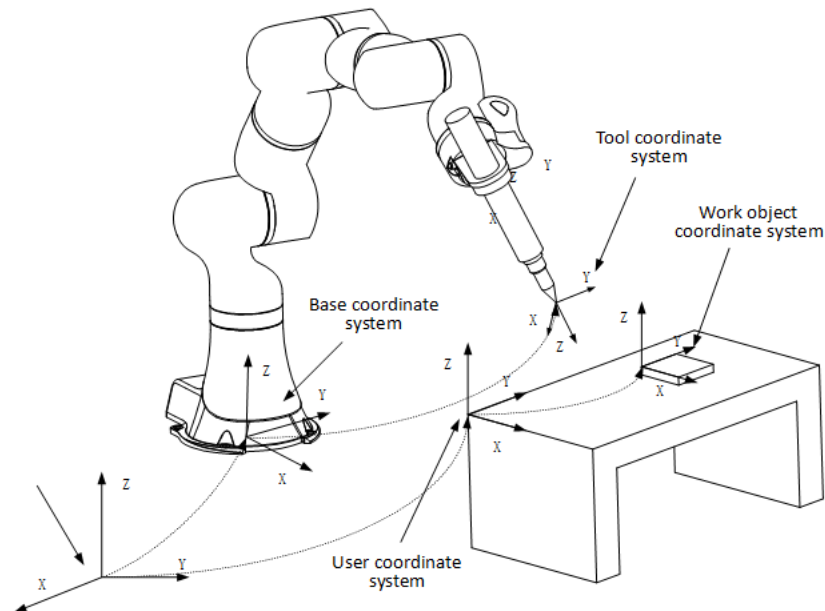
When using **external tools**, the frame definition chain is as follows:
➢ The work object frame is defined relative to the user frame;
➢ The user frame is defined relative to the flange frame.
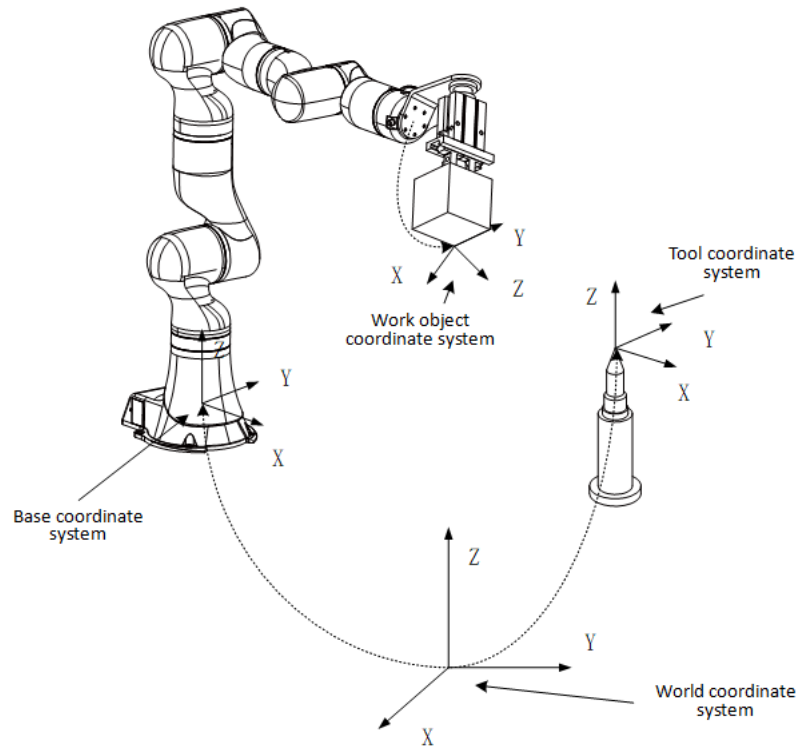
Example

In the variable list:
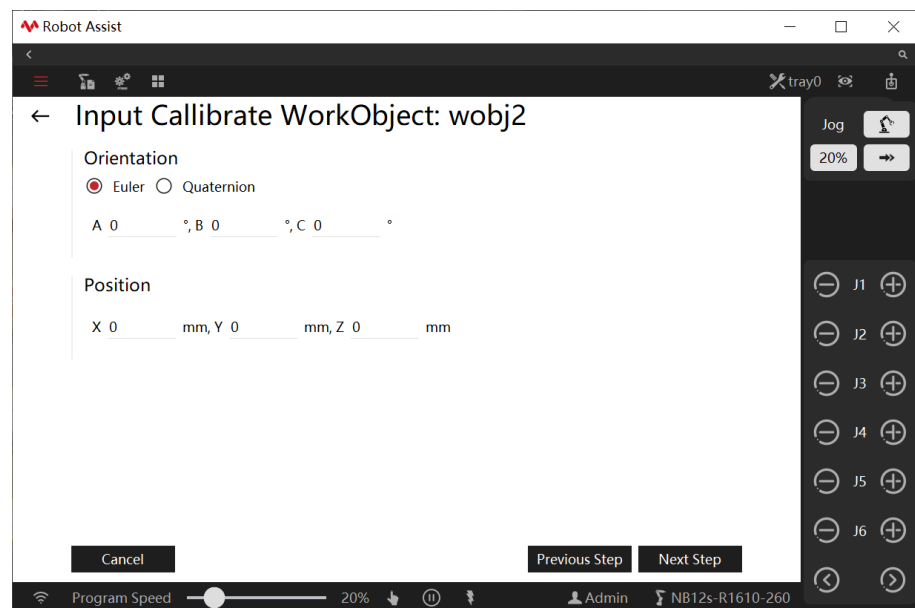
To define the work object named wobj2, where the parameters are:

➢ The work object is mounted on the robot;

➢ The work object frame is fixed and does not move with the external positioner or other robots;

➢ The coordinate values of the origin of the work object frame in the user frame are 300 mm, 600 mm, 200 mm, and the orientation is consistent with the user frame;

The user frame id is 1.

12.1.22 zone

Explanation

The zone variable is used to define how a certain motion ends, or to define the size of the turning zone between two motion trajectories.

For the same target point of robot commands, there are two processing methods in the motion command:

1. When it is processed as a **stop-point**, the robot will move to the target point and reach the target point at a speed of 0 before continuing to execute the next command;

2. When it is processed as a **transition point**, the robot will not move to the target point but will start proceeding to the next target point at a place that is several millimeters away from such a target point. The turning path will deviate from the programmed path. We call the transition area between the two trajectories a turning area. See the following figure for details:
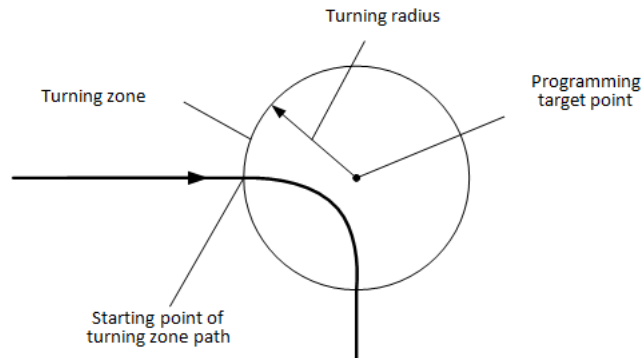
The size of the turning zone cannot exceed half of the path length. If it is exceeded, the system will automatically reduce the turning zone to half the total path length.

The use of turning zones prevents the robot from starting and stopping frequently, significantly reducing the cycle time.

Definition

Joint space trajectories and Cartesian space trajectories define turning zones with different parameters. The variable contains two parts: distance and percent.

distance

Size of turning zone in Cartesian space
Data type: double
It is used for the commands MoveL, MoveC, and MoveT to define the size of the turning zone for Cartesian space trajectories, that is, when the robot moves to a point with a distance of several millimeters to the target point, it starts to move to the next target point, in millimeters. The value ranges from 0 to 200 mm.

percent

Turning percentage
Data type: double
It is used for MoveJ and MoveAbsJ, indicating how far it is to the target angle when starting turning. 100% represents half the value of the entire rotation angle. For command MoveL with pure space-rotation, the parameter Percent is used instead of Distance.

Example

For example, in the variable list, a variable is defined as follows:

Variable Type

zone

Basic Info

Name   zone0

Description   target point

Dimension   [+] [-]   No Array

Select Variable

Current Item: zone0

Edit Value

distance 100

percent 50

A zone variable is defined, in which the size of the Cartesian turning zone is 100 mm and the size of the joint space turning zone is 50%.

Pre-defined variables of turning zone

The system predefines some common turning zone variables, as shown in the following table.

Name	Size of turning zone in Cartesian space	Turning percentage
fine	0 mm	0%
z1	1 mm	1%
z5	5 mm	3%
z10	10 mm	5%
z15	15 mm	8%
z20	20 mm	10%
z30	30 mm	15%
z40	40 mm	20%
z50	50 mm	25%
z60	60 mm	30%
z80	80 mm	40%
z100	100 mm	50%
z150	150 mm	75%
z200	200 mm	100%

Use restrictions

In some special cases, the turning zone will be canceled. The system will report the log "Corner Path Failed".

➤ At least one of the two trajectories is too short (1 mm/0.001 rad);

➤ The two trajectories are nearly parallel and the direction of motion is opposite;

➤ The two trajectories perform pure rotation with the motion axis reversed. Such that only the end-effector axis rotates forward in the previous trajectory, and only the end-effector axis rotates reverse in the latter trajectory.

When a warning for "Turning Zone Canceled" is generated, the program automatically treats the affected command target point as a stop-point.

In addition to the special cases above, all logic commands will cancel the turning zone of the previous motion command.

12.1.23 torqueinfo

Explanation

Describes the forces and torques applied to the robot

It includes joint space torque information and Cartesian space torque information

---

Definition

joint_torque

Data type: Joint space torque information

cart_torque

Data type: Cartesian space torque information

joint_torque.measure_torque

Data type: double array

Information of measured force in the joint space and the torque applied to each axis measured by the force sensor

joint_torque.external_torque

Data type: double array

Information of external force in the joint space, and information of the torque applied to each axis measured by the controller based on the robot model and measured force

cart_torque.m_force

Data type: double array

Force in all directions (xyz) in the Cartesian space

cart_torque.m_torque

Data type: double array

Torque in all directions (xyz) in the Cartesian space

Examples

The following example shows how to use variable torqueinfo:

Example 1

TorqueInfo tmp_info = GetEndtoolTorque(tool1, wobj1)
//Obtain the information architecture of the torque applied to the tool at the end-effector of the robot in the case of tool1 wobj1
…
print(tmp_info.joint_torque.measure_torque)
print(tmp_info.joint_torque.external_torque)
//Print the measured force and external force of each axis
…
print(tmp_info.cart_torque.m_torque)
//Print Cartesian space torque
…
print(tmp_info.cart_torque.m_force[0])
print(tmp_info.cart_torque.m_torque[0])
//Print information of force and torque in X direction

12.1.24 SocketServer

Explanation

A Socket TCP server is established on the controller to listen for connections initiated by external devices as the client. This server is only used to listen for connection requests and multiple connections are supported. When a connection is established, a new SocketConn object is generated for communication.

> **i** Notes
>
> 1、 Do not create (OpenDev) and destroy (CloseDev) server resources too often as it requires time for system resource application and release. It is recommended to keep at least a 500ms time interval between creating and destroying resources, otherwise, system resources will be overloaded and cause problems.

> 2、 This command only creates a server resource object, and the server creation is not completed. The server needs to enter the listening state via OpenDev and SocketAccept.
>
> 3、 The server supports multiple connections.

Definition

ip

Data type: string

The control system uses the ip parameter to match the network interface controller (NIC) and uses the corresponding NIC for network listening. If this parameter is set to "0.0.0.0", it means listening for the connections of all NICs. In most cases, it can be set to "0.0.0.0".

port

Data type: int

Listening port. When an external client initiates a connection, specify the value of the server port set for this purpose.

name

Data type: string

The unique identifier of the server used in the RL program. It is unique within the project and can be shared between multiple tasks without naming conflicts.

Examples

Example 1

SocketServer ss = {"192.168.0.160", 8090, "svr"}   //Only listen for NIC with ip set to 192.168.0.160
SocketConn conn = SocketAccept("svr")

Example 2

SocketServer ss = {"0.0.0.0", 8090, "svr"} //Listen for all NICs of the robot
SocketConn conn = SocketAccept("svr")

12.1.25 SocketConn

Explanation

Socket TCP connection object, used for communication to external devices. There are two types:

1) The robot, as a client, initiates a connection and communication through the object to the TCP server of the external device.

2) The robot acts as a server for communication connections to the counterpart device generated when a connection is initiated by a TCP client of the external device. When multiple TCP client connections are initiated by different external devices, one connection is generated for each connection.

Definition

ip

Data type: string

When the robot is used as a client, this parameter indicates the ip of the external device's server. When the robot is used as a server, this parameter indicates the ip of the external client when a connection is established by the external device.

port

Data type: int

Listening port. When the robot initiates a connection, the server port of the external device should be specified.

name

Data type: string

The unique identifier of the connection used in the RL program. It is unique within the project and can be shared between multiple tasks among connections and between connection and server. Server names should not conflict within the project.

---

cache

Data type: int

Size of the cache, indicating max data received that can be cached. It can be left blank. 1 by default.

suffix

Data type: string

Terminator, indicating the end of a message. It can be left blank. "\r" by default.

attr

Data type: string

Connection attribute.

"incoming": Local server, connected by the opposite-end client. ip and port identify the client information.

"outgoing": Local client, connected to the external server. ip and port identify the opposite-end server connected.

"" and others: Unavailable connection, indicating that the connection has not been opened or unestablished connection has been found.

state

Data type: string

Current communication connection status. Closed: connection closed; established: connection established and working properly.

> **Notes**
>
> 1、 When used as a client, the ip and port information should be set by the user. When used as a server, the ip and port information should be automatically obtained from the accept command. Do not modify these two values easily after the connection is established, unless you are very clear about the use of these two values to avoid errors in program logic and operation.
> 2、 suffix can be reset at any time and can take effect until the next read. Use this feature with caution, as it can cause communication data errors. suffix should be set before communication and should not be modified again.

Examples

Example 1

```
//Server ip "192.168.0.202", port 8090, connection name "clt", cache default to 1, and suffix default to "\r"
SocketConn scnn1 = {"192.168.0.202", 8090, "clt"}
```

Example 2

```
//Server ip "192.168.0.203", port 8091, connection name "clt1", cache 2, and suffix default to "\r"
SocketConn scnn2 = {"192.168.0.203", 8091, "clt1", 2}
```

Example 3

```
//Server ip "192.168.0.204", port 8092, connection name "clt2", cache 2, and suffix "\n"
SocketConn scnn3 = {"192.168.0.204", 8092, "clt2", 2, "\n"}
```

Example 4

```
//Used as server, connection established by the external device
//Server ip "192.168.0.204", port 8092, connection name "clt2", cache 2, and suffix "\n"
SocketConn conn = SocketAccept( "svr1")
Print(conn.ip)    //ip of the external device
Print(conn.port)    //Port of the external device to establish the connection
Print(conn.cache)    //Buffer queue for receiving messages
Print(conn.suffix)    //Sending and receiving suffix
```

12.2 Functions

12.2.1 Functions

Explanation

Use of functions can simplify the code structure, improve the readability and reuse rate of code. The user can define the program segment as a new function that needs to be executed frequently so that it can be conveniently called in the main program at any time.

Function definition

The function is defined as follows:

SCOPE PROC RoutineName()

…

…

//do something

…

…

ENDPROC

Where:

1. SCOPE is the function scope, which supports both the GLOBAL and LOCAL;

2. PROC is the defining keyword of the function;

3. RoutineName is the function name. The naming rules are the same as the variable naming rules. For details, see the Variable naming rules.

Function call

When calling a function, enter the function name directly in the program editor:

RoutineName()

Only other GLOBAL-level functions in this project or LOCAL-level functions in this module file can be called. Recursive calls are not supported. Cross calls between two sub-functions is also not supported.

Calling a function is treated as a separate program command in the compiler.

Notes

> ➤ It is not allowed to define a function in a function.

12.3 Commands

12.3.1 Variable type conversion

12.3.1.1.1 StrToByte

Explanation

StrToByte is used to convert a string with a particular format to byte data.

Return value

Data type: byte

It represents the byte data obtained from the conversion.

Definition

StrToByte (ConStr, [\Hex] | [\Okt] | [\Bin] | [\Char])

ConStr

Data type: string

It represents the string to be converted. If the optional parameter does not exist, it is converted to decimal by default.

\Hex

Identifier, convert by hexadecimal.

\Okt

Identifier, convert by octal.

\Bin

Identifier, converted in binary.

\Char

Identifier, converted according to Ascii character format.

Example

Example 1

VAR byte data
data = StrToByte("10") //10
data = StrToByte("AE" \Hex) //174
data = StrToByte("176" \Okt) //126
data = StrToByte("00001010" \Bin) //10
data = StrToByte("A" \Char) //65

Use restrictions

➢ In the decimal system, the range cannot exceed 0-255, otherwise, an error is reported;
➢ In the hexadecimal system, the range cannot be larger than FF, otherwise, an error is reported;
➢ In the octal system, the range cannot be larger than 377, otherwise, an error is reported;
➢ In the binary system, the range cannot be larger than 11111111, otherwise, an error is reported.

12.3.1.1.2 StrToDouble

Explanation

StrToDouble is used to convert a string to floating-point data.

Return value

Data type: double
Floating-point variable converted from the string.

Definition

StrToDouble (ConStr)

ConStr

Data type: string
It represents the string to be converted.

Example

Example 1

VAR double db_data = StrToDouble("-10") // -10.0
db_data = StrToDouble("45.678")   // 45.678

Use restrictions

➢ If a non-decimal floating-point number is entered, an error is reported.

12.3.1.1.3 StrToInt

Explanation	
	StrToInt is used to convert a string to integer data.

Return value	
	Data type: int
	Integer variable, converted from the string.

Definition	
	StrToInt (ConStr)
ConStr	
	Data type: string
	It represents the decimal numeric string to be converted.

Example	
Example 1	
	VAR int int_data = StrToInt("-10") // -10
	int_data = StrToInt("45678") // 45678

Use restrictions	
	➢ The range of variables to be converted is -2^{31} - 2^{31}. If the range is exceeded, an error is reported.
	➢ If a non-decimal number is entered, an error is reported.

12.3.1.1.4 ByteToStr

Explanation	
	It is used to convert byte-type data to string-type data in a specified format.

Return value	
	Data type: string
	The converted string-type data.

Definition	
	ByteToStr (BitData [\Hex] | [\Okt] | [\Bin] | [\Char])
BitData	
	Data type: byte
	The byte-type data to be converted. Convert by decimal by default.
\Hex	
	Identifier, convert by hexadecimal.
\Okt	
	Identifier, convert by octal.
\Bin	
	Identifier, convert by binary.
\Char	
	Identifier, convert under Ascii character format.

Example	
Example 1	
	VAR byte data1 = 122
	VAR string str1
	str1 = ByteToStr(data1) //"122"

str1 = ByteToStr(data1 \Hex) //"7A"

str1 = ByteToStr(data1 \Okt) //"172"

str1 = ByteToStr(data1 \Bin) //"01111010"

str1 = ByteToStr(data1 \Char) //"z"

Define byte-type variable data1 and assign it with 122, convert data1 to string-type data: 122 by decimal; 7A by hexadecimal;172 by octal; 01111010 by binary; and

z by character.

12.3.1.1.5 DecToHex

Explanation	
	It is used to convert a decimal number to a hexadecimal number.
Return value	
	Data type: string
	It represents the hexadecimal data obtained from the conversion, represented by 0-9, a-f, A-F.
Parameter	
	DecToHex(str)
str	
	Data type: string
	It represents the decimal data to be converted, represented by 0-9.
Use restrictions	
	➢ Data range from 0 to 2147483647 or 0 to 7fffffff.

12.3.1.1.6 DoubleToByte

Explanation	
	It is used to convert a double-type variable or a double array to a byte array.
Return value	
	Data type: byte array
	It represents the byte array obtained from the conversion, each double data is converted to 8 byte-type data.
Parameter	
	DoubleToByte(dou1)
dou1	
	Data type: double
	The double-type variable to be converted.

12.3.1.1.7 DoubleToStr

Explanation	
	It is used to convert a double-type variable to a string.
Parameter	
	DoubleToStr(Val, Dec)

Val

Data type: double

The double-type variable to be converted.

Dec

Data type: int

The number of decimal places to be kept.

Use restrictions

➢ The maximum number of decimal places is 15 digits.

12.3.1.1.8 HexToDec

Explanation

It is used to convert a hexadecimal number to a decimal number.

Return value

Decimal Integer data obtained from the conversion, represented by 0-9.

Parameter

HexToDec(str)

str

Data type: string

The hexadecimal data to be converted, represented by 0-9, a-f, A-F.

Use restrictions

➢ Data range from 0 to 2147483647 or 0 to 7fffffff.

12.3.1.1.9 IntToByte

Explanation

It is used to convert an int-type variable or an int array to a byte array.

Return value

It represents the byte array obtained from the conversion, each int data is converted to four byte data.

Data type: byte array

Parameter

IntToByte(int1)

int1

Data type: int or int array

It represents the int-type variable or int array to be converted.

Use restrictions

➢ Data range from -2147483647 to 2147483647.

12.3.1.1.10 IntToStr

Explanation

It is used to convert integer to string.

Return value

It represents the string obtained from the conversion.

Parameter

int1

IntToStr(int1)

Data type: int
It represents the integer to be converted.

Use restrictions

➢ Data range from -2147483647 to 2147483647.

12.3.2 Motion commands

12.3.2.1 MoveAbsJ

Explanation

MoveAbsJ (Move Absolute Joint) is used to move the robot and the external axis to a position defined by the angle of the axis for rapid positioning or moving the robot to a precise axis angle. All axes move synchronously and the end-effector of the robot moves along an irregular curve. Please be aware of the risk of collision.

The tool parameter used in the MoveAbsJ command would not affect the end position of the robot, but the tool parameters are still being used by the controller for dynamics calculations.

Parameter

TojointPos

MoveAbsJ ToJointPos, Speed, Zone, Tool, [Wobj]
TThe parameter in [ ] is optional and can be omitted.

Target joint angle (*To Joint Position*)
Data type: jointtarget
The target angle and position value of the robot and the external axis.

Speed

Motion Speed
Data type: speed
It is used to specify the motion speed of the robot when it executes MoveAbsJ, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.

Zone

Turning Zone
Data type: zone
It is used to define the size of the turning zone for the current trajectory.

Tool

Data type: tool
The tool used when executing the trajectory.
The command MoveAbsJ calculates the motion speed and the size of the turning zone using the tool's TCP data.

[Wobj]

Work Object
Data type: wobj
The work object used when executing this trajectory.
When the tool is installed on the robot, this parameter can be ignored;

When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
	The following are some examples for MoveAbsJ:
Example 1	
	MoveAbsJ j10, v500, fine, tool1
	The robot moves along an irregular path at a velocity of v500 to the absolute joint angle as defined by j10 using tool1, with a turning zone of 0.
Example 2	
	MoveAbsJ startpoint, v1000, z100, gripper, phone
	The robot moves along the irregular path to the absolute joint angle defined by the startpoint at a velocity of v1000 in the work object frame by using the tool gripper, with a turning zone of 100 mm.

12.3.2.2 MoveJ

Explanation	
	MoveJ (Move The Robot By Joint Motion) is used to move the robot from one point to another when the motion trajectory of the robot end-effector is not required. All axes move synchronously and the end-effector of the robot moves along an irregular curve. Please be aware of the risk of collision. The biggest difference between the commands MoveJ and MoveAbsJ is that the given target point format is different. The target point of MoveJ is the spatial pose of the tool (TCP) rather than the joint axis angle.

Parameter	
	MoveJ ToPoint, Speed, Zone, Tool, [Wobj]
	The parameter in [ ] is optional and can be omitted.
ToPoint	
	Target pose (To Point)
	Data type: robtarget
	The target position described in the Cartesian space.
Speed	
	Motion Speed
	Data type: speed
	It is used to specify the motion speed of the robot when it executes MoveJ, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.
Zone	
	Turning Zone
	Data type: zone
	It is used to define the size of the turning zone for the current trajectory.
Tool	
	Data type: tool
	The tool used when executing the trajectory.
	The command MoveJ calculates the motion speed and the size of the turning zone using the tool's TCP data.
[Wobj]	
	Work Object
	Data type: wobj
	The work object used when executing this trajectory.
	When the tool is installed on the robot, this parameter can be ignored;
	When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
	The following are some examples for MoveJ:
Example 1	
	MoveJ p30, v100, z50, tool1 The robot moves the TCP along the irregular path to the target point defined by p30 at a velocity of v100 using the tool1, with a turning zone of 50 mm.
Example 2	
	MoveJ endpoint, v500, z50, gripper, wobj2
	The robot moves the TCP along the irregular path to the target point defined by the endpoint at a velocity of v500 in the work object frame wobj2 by using the gripper, with a turning zone of 50 mm.

12.3.2.3 MoveL

Explanation	
	MoveL (Move Line) is used to move the TCP along a straight line to a given target position. When the starting and ending orientations are different, the orientation will be rotated synchronously with the position to the endpoint. Since the translation and rotation speeds are specified separately, the final motion time of the MoveL command depends on the change time of orientation, position, and elbow (whichever is longer) in order not to exceed the specified speed limit. Therefore, when performing certain trajectories (for example, small displacements but with large changes in orientation), if the robot is moving at a significantly slower or faster speed, please check whether the rotation speed setting is reasonable. When you need to keep the TCP stationary by only adjusting the tool orientation, you can achieve this by specifying the starting point and endpoint for MoveL with the same position but with a different orientation.

Parameter	
	MoveL ToPoint, Speed, Zone, Tool, [Wobj]
	The parameter in [ ] is optional and can be omitted.
ToPoint	
	Target pose (*To Point*) Data type: robtarget The target position described in the Cartesian space.
Speed	
	Motion Speed Data type: speed It is used to specify the motion speed of the robot when it executes MoveL, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.
Zone	
	Turning Zone Data type: zone It is used to define the size of the turning zone for the current trajectory.
Tool	
	Data type: tool The tool used when executing the trajectory. The speed in the command refers to the tool's TCP speed and rotation speed.
[Wobj]	
	Work Object Data type: wobj The work object used when executing this trajectory. When the tool is installed on the robot, this parameter can be ignored;

When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
	The following are some examples for MoveL:
Example 1	
	MoveL p10, v1000, z50, tool0
	The robot moves the TCP along the straight path to the target point defined by p10 at a velocity of v1000 using the tool0, with a turning zone of 50 mm
Example 2	
	MoveL endpoint, v500, z50, gripper, wobj2
	The robot moves the TCP along the straight path to the target point defined by the endpoint at a velocity of v500 in the work object frame wobj2 by using the gripper, with a turning zone of 50 mm.

12.3.2.4 MoveC

Explanation	
	MoveC (Move Circle) is used to move the TCP along the arc through the middle auxiliary point to the given target position.
	When the starting and ending orientations are different, the orientation will rotate synchronously as the position moves to the end position. The orientation at the auxiliary point does not affect the arc motion process.
	Since the translation and rotation speeds are specified separately, the final motion time of the MoveC command depends on the change time of orientation, position, and elbow (whichever is longer) in order not to exceed the specified speed limit. Therefore, in certain trajectories (for example, small displacements but with large changes in orientation), if the robot is moving at a significantly slower or faster speed, please check whether the rotation speed setting is reasonable.

Parameter	
	MoveC AuxPoint, ToPoint, Speed, Zone, Tool, [Wobj]
	The parameter in [ ] is optional and can be omitted.
AuxPoint	
	Auxiliary Point
	Data type: robtarget
	The position of the auxpoint described in the Cartesian space is used to determine the size of the arc and the direction of motion. The orientation of this point does not affect the execution of the final trajectory.
ToPoint	
	Target pose (To Point)
	Data type: robtarget
	The target position described in the Cartesian space.
Speed	
	Motion Speed
	Data type: speed
	It is used to specify the motion speed of the robot when it executes MoveC, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.
Zone	
	Turning Zone
	Data type: zone
	It is used to define the size of the turning zone for the current trajectory.

Tool	
	Data type: tool
	The tool used when executing the trajectory. The speed in the command refers to the tool's TCP speed and rotation speed.
[Wobj]	
	Work Object
	Data type: wobj
	The work object used when executing this trajectory.
	When the tool is installed on the robot, this parameter can be ignored;
	When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
	The following are some examples for MoveC:
Example 1	
	MoveC p10, p20, v1000, z50, tool0
	The robot moves the TCP along the arc, passing through the p10 point to the target point defined by p20 at a velocity of v1000 using the tool0, with a turning zone of 50 mm.
Example 2	
	MoveC auxpoint, endpoint, v500, z50, gripper, wobj2
	The robot moves the TCP along the arc, passing through auxpoint to the target point defined by the endpoint at a velocity of v500 in the work object frame wobj2 by using the gripper, with a turning zone of 50 mm.

12.3.2.5 MoveT

Explanation	
	MoveT (Move trochoid) is used to move the TCP to a given target position through rotary stepping with a trochoid passing through auxiliary points.
	When the starting and ending orientations are different, the pose will rotate synchronously as the position moves to the end position. The orientation at the auxiliary point does not affect the spiral motion process.
	Since the translation and rotation speeds are specified separately, the final motion time of the MoveT command depends on the change time of orientation, position, and elbow (whichever is longer) in order not to exceed the specified speed limit. Therefore, in certain trajectories (for example, small displacements but with large changes in orientation), if the robot is moving at a significantly slower or faster speed, please check whether the rotation speed setting is reasonable.

Parameter	
	MoveC AuxPoint, ToPoint, Radius, Step, Speed, Zone, Tool, [Wobj]
	The parameter in [ ] is optional and can be omitted.
AuxPoint	
	Auxiliary Point
	Data type: robtarget
	The position of the auxpoint described in the Cartesian space is used to determine the size of the arc and the direction of motion. The orientation of this point does not affect the execution of the final trajectory.
ToPoint	
	Target pose (To Point)
	Data type: robtarget
	The target position described in the Cartesian space.

Radius

Cycloid radius

Data type: double

Radius of trochoid advance, in mm

Step

Step length

Data type: double

Step length of trochoid advance, in mm

Speed

Motion Speed

Data type: speed

It is used to specify the motion speed of the robot when it executes MoveT, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.

Zone

Turning Zone

Data type: zone

It is used to define the size of the turning zone for the current trajectory.

Tool

Data type: tool

The tool used when executing the trajectory. The speed in the command refers to the tool's TCP speed and rotation speed.

[Wobj]

Work Object

Data type: wobj

The work object used when executing this trajectory.

When the tool is installed on the robot, this parameter can be ignored;

When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example

The following are some examples for MoveT:

Example 1

MoveT p10, p20, 150, 50, v1000, z50, tool0

With tool0, the robot TCP draws a trochoid that passes point p10 in an arc at a velocity of v1000.

With a trochoid radius of 150 mm and a step of 50 mm, the TCP finally moves to the target position defined by p20, with a turning zone size of 50 mm.
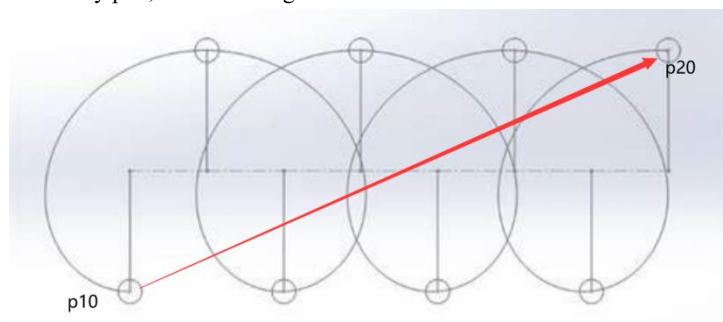
12.3.2.6 SearchL

Explanation

SearchL (Search Liner) is used to search the position when moving the TCP along a straight line. During the movement, the robot will monitor a digital input (DI) signal. When the signal status monitored matches the trigger mode, the robot immediately reads the current position.

The command can be used when the tool fixed to the manipulator is a probe used for surface

detection. Use the SearchL command to obtain the outline coordinates of the work object.

The command can only be used for motion tasks.

Parameter

SearchL [action,] di, [trigger_mode,] save_rob, target_rob, Speed, Tool [,Wobj]

The parameter in [ ] is optional and can be omitted.

action

Action after triggering DI

Data type: keyword

Blank: no stop

\Stop: quick stop, which may cause the robot to deviate from the path. But the robot stops quickly.

Only available when the speed is below v100

\PStop: planned stop. The robot will stop on the specified path, without speed limits

di

Data type: DI signal

SearchL command triggers signal of specified action, and user-defined DI signal is used

trigger_mode

DI signal trigger mode

Data type: keyword

Blank: posedge triggering by default

\Flanks: edge triggering (posedge/negedge)

\Posflank: posedge triggering

\Negflank: negedge triggering

\Highlevel: high-level triggering

\Lowlevel: low-level triggering

save_rob

Data type: robtarget

Save the point position of the position data when the robot triggers the signal

target_rob

Data type: robtarget

Target point position of linear motion

Speed

Motion Speed

Data type: speed

It is used to specify the motion speed of the robot when it executes Search, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.

Tool

Data type: tool

The tool used when executing the trajectory. The speed in the command refers to the tool's TCP speed and rotation speed.

[Wobj]

Work Object

Data type: wobj

The work object used when executing this trajectory.

When the tool is installed on the robot, this parameter can be ignored;

When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example

The following are some examples for SearchL:

Example 1

SearchL di0, save_rob, target_rob, v500, tool0

The robot uses tool0 and TCP moves towards target_rob in a straight line at v500. If di0 jumps to high during the motion, the robot's coordinate information at the time of the signal jump is recorded in save_rob.

Example 2

SearchL \PStop, di0, \Lowlevel, save_rob, target_rob, v500, tool0

The robot uses tool0 and TCP moves towards target_rob in a straight line at v500. If di0 is low during the motion, the robot will immediately have a planned stop and record the robot's coordinate information in save_rob when the signal is detected to be low.

12.3.2.7 SearchC

Explanation

SearchC (Search Circle) is used to search for a position when moving the TCP along a circle.
During the movement, the robot will monitor a digital input (DI) signal. When the signal status monitored matches the trigger mode, the robot immediately reads the current position.
The command can be used when the tool fixed to the manipulator is a probe used for surface detection. Use SearchC command
to obtain the outline coordinates of the work object.
The command can only be used for motion tasks.

Parameter

SearchC [action,] di, [trigger_mode,] save_rob, aux_rob, target_rob, Speed, Tool [,Wobj]
The parameter in [ ] is optional and can be omitted.

action

Action after triggering DI
Data type: keyword
Blank: no stop
\Stop: quick stop, which may cause the robot to deviate from the path. But the robot stops quickly.
Only available when the speed is below v100
\PStop: planned stop. The robot will stop on the specified path, without speed limits

di

Data type: DI signal
SearchC command triggers signal of specified action, and user-defined DI signal is used

trigger_mode

DI signal trigger mode
Data type: keyword
Blank: posedge triggering by default
\Flanks: edge triggering (posedge/negedge)
\Posflank: posedge triggering
\Negflank: negedge triggering
\Highlevel: high-level triggering
\Lowlevel: low-level triggering

save_rob

Data type: robtarget
Save the point position of the position data when the robot triggers the signal

aux_rob

Data type: robtarget
Auxiliary point during circular motion

target_rob

Data type: robtarget
Target point position of circular motion

Speed

Motion Speed
Data type: speed

	It is used to specify the motion speed of the robot when it executes Search, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis.
Tool	Data type: tool The tool used when executing the trajectory. The speed in the command refers to the tool's TCP speed and rotation speed.
[Wobj]	*Work Object* Data type: wobj The work object used when executing this trajectory. When the tool is installed on the robot, this parameter can be ignored; When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
	The following are some examples for SearchC:
Example 1	SearchC di0, save_rob, aux_rob, target_rob, v500, tool0 The robot uses tool0 and TCP moves at a speed of v500 towards target_rob in a circle after passing auxiliary point aux_rob. If di0 jumps to high during the motion, the robot's coordinate information at the time of the signal jump is recorded in save_rob.
Example 2	SearchC \PStop, di0, \Flanks, save_rob, target_rob, v500, tool0 The robot uses tool0 and TCP moves at a speed of v500 towards target_rob in a straight line after passing auxiliary point aux_rob. If di0 jumps from low to high or from high to low during the motion, the robot immediately has a planned stop and the robot's coordinate information at the time of the signal jump is recorded in save_rob.

12.3.3 Trigger command

12.3.3.1 TrigIO

Explanation	
	TriggIO is used to set a trigdata as an output I/O trigger during the motion. Digital output DO and digital group output GO are supported.

Definition	

	TrigIO TrigData,Distance,RefStart,SignalName,Value
	Parameter:
	TrigData (data type: trigdata) is a variable used to store the trigger data set by this TrigIO.
	Distance (data type: double, and non-negative (negative numbers are treated as 0)) defines the location offset of the trigger event on the path. Whether the location offset is relative to the path start or end is defined by RefStart;
	RefStart (data type: bool) defines whether the trigger position is relative to the start point (true) or the end point (false).
	SignalName (data type: signaldo or signalgo) is the signal name of the digital output or digital group output associated with this defined IO event, which must be an output signal that has been set correctly; //add
	Value (data type: bool or int) defines the target value of the output signal when an IO event is triggered. The data type of the given value should match the SignalName type.

Example	
Example 1	Refer to the TrigL example;

12.3.3.2 TrigReg

Explanation	
	TrigReg is used to set a trigdata to modify the register value during the motion; register types supported include int16, bool, float, and bit.

Definition	
	TrigReg TrigData,Distance,RefStart,RegName,Value
	Parameter:
	TrigData (data type: trigdata) is a variable used to store the trigger data set by this TrigIO.
	Distance (data type: double, and non-negative (negative numbers are treated as 0)) defines the location offset of the trigger event on the path. Whether the location offset is relative to the path start or end is defined by RefStart;
	RefStart (data type: bool) defines whether the trigger position is relative to the start point (true) or the end point (false).
	RegName refers to the register name, and the data type is not available. Note: Registers can not be created in RL. The user needs to create new registers through "Robot -> Communication -> Register";
	Value (data type int16, bool, float, or bit) defines the target value of the register when a register modification event is triggered. The data type of the given value should match the RegName type; if the value specified by the user mismatches with the register type, the type will be transformed automatically.

Example	
Example 1	
	Refer to the TrigL example;

12.3.3.3 TrigL

Explanation	
	Like MoveL, TrigL is a command to perform linear motion in space. The difference is that TrigL can perform predefined operations at several specified positions during the motion; the two commands are the same in the number and meaning of other parameters.

Definition	
	TrigL ToPoint,Speed,Trigger,Zone,Tool,[Wobj]
	Parameter:
	ToPoint, or target pose (data type: robtarget), describes the target pose in Cartesian space;
	Speed (type: speed) is used to specify the motion speed of the robot when it executes MoveL, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis;
	Trigger, or trigger condition and action, (type: trigdata; trigdata) must be the trigdata processed with TrigX command, otherwise, the compiler will report an error when coming to this line.
	Zone, or Turning zone (type: zone) is used to define the size of the turning zone for the current trajectory;
	Tool (type: tool);
	[Wobj], or work object (type: wobj) refers to the work object used when executing this trajectory.
	When the tool is installed on the robot, this parameter can be ignored; When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example

Example 1

> VAR trigdata tc1
>
> VAR trigdata tc2
>
> VAR trigdata tc3
>
> ...
>
> //Set tc1, tc2, tc3
>
> TrigIO tc1,0,true,do2,true
>
> TrigIO tc2,60,false,do2,false
>
> TrigReg tc3,80,true,r0,false //r0 is a bool type register
>
> ...
>
> //Motion
>
> MoveL p1,v500,z50,tool1
>
> TrigL p2,v500,tc1,z50,tool1
>
> TrigL p3,v500,tc2,fine,tool1
>
> TrigL p4,v500,tc3,fine,tool1
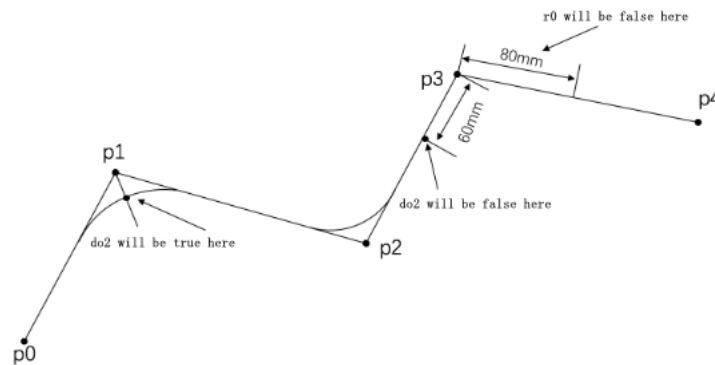
12.3.3.4 TrigC

Explanation

> TriggC is similar to MoveC in that it is a command to execute circular motion. The difference is that TriggC can perform predefined operations at several specified positions during the motion; the two commands are the same in the number and meaning of other parameters.

Definition

> TrigC AuxPoint,ToPoint,Speed,Trigger,Zone,Tool,[Wobj]
>
> Parameter:
>
> AuxPoint, or Auxiliary Point (data type: robtarget), describes the target pose in Cartesian space;
>
> ToPoint, or target pose (data type: robtarget), describes the target pose in Cartesian space;
>
> Speed (type: speed) is used to specify the motion speed of the robot when it executes MoveL, including the translation speed of the robot end-effector, the rotation speed, and the motion speed of the external axis;
>
> Trigger, or trigger condition and action, (type: trigdata; trigdata) must be the trigdata processed with TrigX command, otherwise, the compiler will report an error when coming to this line.
>
> Zone, or Turning zone (type: zone) is used to define the size of the turning zone for the current trajectory;
>
> Tool (type: tool);
>
> [Wobj], or work object (type: wobj) refers to the work object used when executing this trajectory.

When the tool is installed on the robot, this parameter can be ignored; When using external tools, this parameter must be specified, and the robot will calculate the motion speed and the size of the turning zone by using the data saved in wobj.

Example	
Example 1	

VAR trigdata tc1

...

//Set tc1

TrigIO tc1,0,true,do2,true

...

//Motion

MoveL p1,v500,z50,tool1

TrigC p2,p3,v500,tc1,fine,tool1
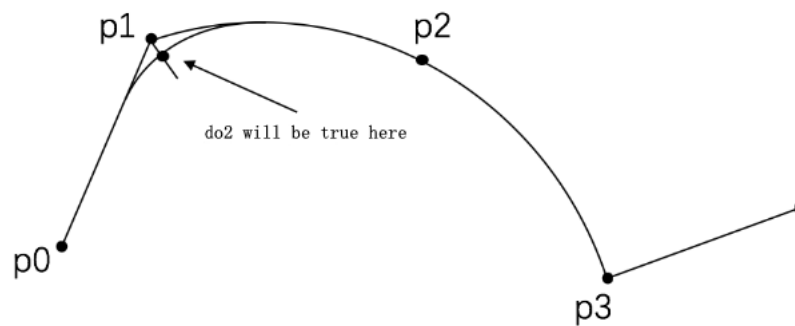
12.3.4 Force control commands

12.3.4.1 CalibSensorError

Explanation	

It is used to clear the six-dimensional force measurement on the end-effector and use the current measurement as zero point.

Definition	

CalibSensorError

No parameters, and can be used directly.

Example	
Example 1	

FcInit Tool1, Wobj0, 0

FcStart

SetSensorUseType 1

CalibSensorError

Set software zeroing for the six-dimensional force measurement and clear the zero.

Use restrictions	

➢ This interface can only be called after executing SetSensorUseType 1, i.e. set software zeroing for the six-dimensional force measurement. If not, the end-effector six-dimensional measurement will not be cleared successfully.

12.3.4.2 FcInit

Explanation	
	It is used for initialization before the force control is enabled, such as setting the work object, tool, and force control frame.

Definition	
	FcInit Tool, Wobj, ForceFrameRef
Tool	
	Data type: pose
	The tool used for force control. The origin of the force control frame is the TCP of the tool (the orientation is the same as the orientation of the frame selected in the third parameter). Note that all adapter flanges used need to be included in the definition of the tool.
Wobj	
	Data type: pose
	The work objects used for force control. Many force control functions are defined relative to the work object frame, such as the orientation of the force control frame, the search mode, and termination conditions. This parameter is Wobj0 by default.
ForceFrameRef	
	Data type: int
	It is used to define the frame to which the force control frame is relative. It supports:
	0: World frame
	1: Work object frame
	2: Tool frame
	The default value is the work object frame (0).

Example	
Example 1	
	FcInit Tool1, Wobj0, 0
	Initialize force control, and define the tool1 and work object wobj0 used when force control is enabled, and the definition of force control frame in relative to the world frame.

Use restrictions	
	➢ FcInit is not allowed to be called again between FcInit and FcStop.

12.3.4.3 SetControlType

Explanation	
	It is used to set the impedance control type.

Definition	
	SetControlType ctrl_type
ctrl_type	
	Data type: int
	Impedance control type, including:
	0: Joint impedance
	1: Cartesian impedance

Example

Example 1

> FcInit Tool1, Wobj0, 0
> SetControlType 0
> Set joint impedance as the impedance control mode after executing FcInit.

Use restrictions

> ➢ The impedance type can only be set after executing FcInit and before executing FcStart.

12.3.4.4 SetSensorUseType

Explanation

> It is used to set how to use the six-dimensional force measurement on the end-effector.

Definition

> SetSensorUseType sensor_use_type

sensor_use_type

> Data type: int
> Sensor usage method, supporting:
> 0: Dynamic compensation
> 1: Software zeroing

Example

Example 1

> FcInit Tool1, Wobj0, 0
> FcStart
> SetSensorUseType 0
> Set dynamic compensation as the six-dimensional force measurement, which means the value can be used directly without zeroing.

Use restrictions

> ➢ This interface can only be called after executing FcStart and before executing FcStop. If not, the use method of the end-effector six-dimensional measurement will not be set successfully.

12.3.4.5 SetCartNSStiff

Explanation

> It is used to set the null-space impedance stiffness.

Definition

> SetCartNSStiff cart_ns_stiff

cart_ns_stiff

> Data type: double
> Cartesian null-space impedance stiffness, range: 0~4, in N.m/rad.

Example

Example 1

> FcInit Tool1, Wobj0, 0
> SetControlType 1
> SetCartNSStiff 2
> Set Cartesian impedance as the impedance control mode and the null-space impedance stiffness as 2.

Use restrictions

> ➤ This interface can only be called after executing SetControlType 1, that is, setting Cartesian impedance as the impedance control mode.
>> If not, the null-space impedance parameters will not be set successfully.

12.3.4.6 SetJntCtrlStiffVec

Explanation	
	It is used to set the joint impedance stiffness.

Definition	
	SetJntCtrlStiffVec jnt1_stiff, jnt2_stiff, jnt3_stiff, jnt4_stiff, jnt5_stiff, jnt6_stiff, jnt7_stiff
jnt1_stiff	
	Data type: double
	Impedance stiffness of joint 1, range: 0~1500, in Nm/rad.
Jnt2_stiff	
	Data type: double
	Impedance stiffness of joint 2, range: 0~1500, in Nm/rad.
Jnt3_stiff	
	Data type: double
	Impedance stiffness of joint 3, range: 0~1500, in Nm/rad.
Jnt4_stiff	
	Data type: double
	Impedance stiffness of joint 4, range: 0~1500, in Nm/rad.
Jnt5_stiff	
	Data type: double
	Impedance stiffness of joint 5, range: 0~100, in Nm/rad.
Jnt6_stiff	
	Data type: double
	Impedance stiffness of joint 6, range: 0~100, in Nm/rad.
Jnt7_stiff	
	Data type: double
	Impedance stiffness of joint 7, range: 0~100, in Nm/rad.

Example	
Example 1	
	FcInit Tool1, Wobj0, 0
	SetControlType 0
	SetJntCtrlStiffVec 1500,1500, 1500,1500,100,100,100
	Set the joint impedance as the impedance control mode and the impedance stiffness of joints 1~7 as 1500, 1500, 1500, 1500, 100, 100, 100, respectively.

Use restrictions	
	➤ This interface can only be called after executing SetControlType 0, that is, setting joint impedance as the impedance control mode.
	If not, the joint impedance parameters will not be set successfully.

12.3.4.7 SetCartCtrlStiffVec

Explanation	
	It is used to set the Cartesian impedance stiffness.

Definition	
	SetCartCtrlStiffVec trans_stiff_x, trans_stiff_y, trans_stiff_z, rot_stiff_x, rot_stiff_y, rot_stiff_z
trans_stiff_x	

Data type: double

Cartesian impedance force stiffness in the x-direction, range: 0~1500, in N/m.

trans_stiff_y

Data type: double

Cartesian impedance force stiffness in the y-direction, range: 0~1500, in N/m.

trans_stiff_z

Data type: double

Cartesian impedance force stiffness in the z-direction, range: 0~1500, in N/m.

rot_stiff_x

Data type: double

Cartesian impedance torque stiffness in the x-direction, range: 0~100, in N.m/rad.

rot_stiff_y

Data type: double

Cartesian impedance torque stiffness in the y-direction, range: 0~100, in N.m/rad.

rot_stiff_z

Data type: double

Cartesian impedance torque stiffness in the z-direction, range: 0~100, in N.m/rad.

Example

Example 1

FcInit Tool1, Wobj0, 0
SetControlType 1
SetCartCtrlStiffVec 1000, 1000, 1000, 100, 100, 100

Set Cartesian impedance as the impedance control mode and the impedance force stiffness in x/y/z direction as 1000, and the impedance torque stiffness as 100.

Use restrictions

> This interface can only be called after executing SetControlType 1, that is, setting Cartesian impedance as the impedance control mode.
> If not, the Cartesian impedance parameters will not be set successfully.

12.3.4.8 SetJntTrqDes

Explanation

Set the desired torque of the joint.

Definition

SetJntTrqDes tau_d1,tau_d2,tau_d3,tau_d4,tau_d5,tau_d6,tau_d7

tau_d1

Data type: double

The desired torque of joint 1, range: 0~20, in N.m.

tau_d2

Data type: double

The desired torque of joint 2, range: 0~20, in N.m.

tau_d3

Data type: double

The desired torque of joint 3, range: 0~20, in N.m.

tau_d4

Data type: double

The desired torque of joint 4, range: 0~20, in N.m.

tau_d5

Data type: double

The desired torque of joint 5, range: 0~20, in N.m.

tau_d6

Data type: double

The desired torque of joint 6, range: 0~20, in N.m.

tau_d7

Data type: double

The desired torque of joint 7, range: 0~20, in N.m.

Example

Example 1

FcInit Tool1, Wobj0, 0

SetControlType 0

FcStart

SetJntTrqDes 5,5,5,5,5,5,5

FcStop

Set the desired torque of all joints to 5N.m.

Use restrictions

➢ This interface can only be called after executing FcStart and before executing FcStop. If not, the desired joint torque will not be set successfully.

12.3.4.9 SetCartForceDes

Explanation

It is used to set the desired Cartesian force/torque.

Definition

SetCartForceDes force_x, force_y, force_z, torque_x, torque_y, torque_z

force_x

Data type: double

Desired Cartesian force in the x-direction, in N.

force_y

Data type: double

Desired Cartesian force in the y-direction, in N.

force_z

Data type: double

Desired Cartesian force in the z-direction, in N.

torque_x

Data type: double

Desired Cartesian torque in the x-direction, range: 0-20, in N.m.

torque_y

Data type: double

Desired Cartesian torque in the y-direction, range: 0-20, in N.m.

torque_z

Data type: double

Desired Cartesian torque in the z-direction, range: 0-20, in N.m.

Example

Example 1

FcInit Tool1, Wobj0, 0

SetControlType 1

FcStart

SetCartForceDes 0,0,5,0,0,0

FcStop

Set the desired Cartesian force/torque. Set the desired force in the z-direction to 5N.

Use restrictions

> ➤ This interface can only be called after executing FcStart and before executing FcStop. If not, the desired Cartesian force/torque will not be set successfully.

12.3.4.10 SetSineOverlay

Explanation

It is used to set the sine overlay rotating around a single axis.

Definition

SetSineOverlay line_dir, amplify, frequncy, phase, bias

line_dir

Data type: int
line_dir: overlay reference axis, supporting:
0: x-axis as the reference direction
1: y-axis as the reference direction
2: z-axis as the reference direction

amplify

Data type: double
Overlay amplitude, range: 0~10, in N.m.

frequncy

Data type: double
Overlay frequency, range: 0~5, in Hz.

phase

Data type: double
Overlay phase, range: 0~3.14, in rad.

bias

Data type: double
Overlay offset, range: 0~10, in N.m.

Example

Example 1

FcInit Tool1, Wobj0, 0
SetControlType 1
SetSineOverlay 0, 10, 5, 3.14, 2
Set rotary overlay around x-axis (0), amplitude: 10N.m, frequency: 5Hz, phase: 3.14, and offset: 2N.m.

Use restrictions

> ➤ This interface can only be called after executing SetControlType 1, that is, setting Cartesian impedance as the impedance control mode, and before executing StartOverlay. If not, the sine overlay will not be set successfully.

12.3.4.11 SetLissajousOverlay

Explanation

It is used to set the Lissajous overlay within a plane.

Definition

SetLissajousOverlay plane, amplify_one, frequncy_one, amplify_two, frequncy_two, phase_diff

plane

Data type: int

Overlay reference plane, supporting:

0: XY plane as the reference plane

1: XZ plane as the reference plane

2: YZ plane as the reference plane

amplify_one

Data type: double

amplify_one: The amplitude of overlay in Direction 1, range: 0~10, in N.m.

frequncy_one

Data type: double

frequncy_one: The frequency of overlay in Direction 1, range: 0~5, in Hz.

amplify_two

Data type: double

amplify_two: The amplitude of overlay in Direction 2, range: 0~10, in N.m.

frequncy_two

Data type: double

frequncy_two: The frequency of overlay in Direction 2, range: 0~5, in Hz.

phase_diff

Data type: double

phase_diff: The phase deviation between overlays in two directions, range: 0~3.14, in rad.

Example

Example 1

FcInit Tool1, Wobj0, 0

SetControlType 1

SetLissajousOverlay 0, 5, 2.5, 10, 5, 3.14

Set Lissajous overlay within the xy plane (0). The amplitude and frequency are 5N.m and 2.5Hz in the x-direction, and 10N.m and 5Hz in the y-direction. The phase deviation between the y-direction and x-direction is 3.14.

Use restrictions

➢ This interface can only be called after executing SetControlType 1, that is, setting Cartesian impedance as the impedance control mode, and before executing StartOverlay. If not, the overlay will not be set successfully.

12.3.4.12 SetLoad

Explanation

It is used to set the load information used by the force control module.

Definition

SetLoad m,rx,ry,rz,Ixx,Iyy,Izz

m

Data type: double

Load mass, unit: kg

rx

Data type: double

The position of the load's center of mass on the x-axis of the flange frame, in mm.

ry

Data type: double

rz

The position of the load's center of mass on the y-axis of the flange frame, in mm.

Data type: double

The position of the load's center of mass on the z-axis of the flange frame, in mm.

Ixx

Data type: double

The inertia of the load's center of mass along the x-axis, in kg*mm^2.

Iyy

Data type: double

The inertia of the load's center of mass along the y-axis, in kg*mm^2.

Izz

Data type: double

The inertia of the load's center of mass along the z-axis, in kg*mm^2.

Example

Example 1

FcInit Tool1, Wobj0, 0

FcStart

SetLoad 1,0,0,10,0.001,0.001,0.0001

Set the end-effector load as follows: the mass is 1kg, the component of the center of mass in the flange frame is 0, 0, and 10 mm, and the inertia of the load relative to the load's center of mass frame is 0.001kg*mm^2, 0.001kg*mm^2, and 0.0001kg*mm^2, respectively.

Use restrictions

➢ The interface can only be called after executing FcStart. If not, the load parameters will not be set successfully.

12.3.4.13 FcStart

Explanation

It is used to enable force control. It switches the robot from pure position control to force control.

Definition

FcStart

No parameters, and can be used directly.

Example

Example 1

FcInit Tool1, Wobj0, 0

FcStart

Enable force control through FcStart after executing FcInit. The robot is now in force control mode.

Use restrictions

➢ This interface is called after executing FcInit. Before calling the command, the robot mechanical zero, force sensor zero, and load information should be set correctly, and the body parameters are identified correctly. Otherwise, the effectiveness of the force control function will be affected or even disabled.

12.3.4.14 FcStop

Explanation

It is used to stop force control. The robot will switch from force control to position control.
Executing this command will automatically stop all overlays.

Definition	
	FcStop
	No parameters, and can be used directly.

Example	
Example 1	
	FcInit Tool1, Wobj0, 0
	FcStart
	FcStop
	It is used to stop force control. The robot will switch from force control to position control. Executing this command clears all force control states.

Use restrictions	
	➤ This interface is called after executing FcStart, and it will clear the force control state, such as force control load information, impedance parameters, overlay, desired force, etc. To enable force control again, FcInit should be executed again.

12.3.4.15 StartOverlay

Explanation	
	Enable the overlay set before.

Definition	
	StartOverlay
	No parameters, and can be used directly.

Example	
Example 1	
	FcInit Tool1, Wobj0, 0
	SetControlType 1
	SetSineOverlay 0, 10, 5, 3.14, 2
	SetLissajousOverlay 0, 5, 2.5, 10, 5, 3.14
	FcStart
	StartOverlay
	Start the superposition of overlays set before. In the example, these overlays include the sine overlay around the x-axis and the Lissajous overlay within xy plane.

Use restrictions	
	➤ The interface can only be called after executing FcStart. If not, the sine overlay will not be set successfully.

12.3.4.16 PauseOverlay

Explanation	
	Pause the overlay.

Definition	
	PauseOverlay
	No parameters, and can be used directly.

Example	

Example 1

> FcInit Tool1, Wobj0, 0
> SetControlType 1
> SetSineOverlay 0, 10, 5, 3.14, 2
> FcStart
> StartOverlay
> PauseOverlay
> Pause the overlay.

Use restrictions

> ➢ The interface can only be called after executing StartOverlay.

12.3.4.17 RestartOverlay

Explanation

> Restart the paused overlays.

Definition

> RestartOverlay
> No parameters, and can be used directly.

Example

Example 1

> FcInit Tool1, Wobj0, 0
> SetControlType 1
> SetSineOverlay 0, 10, 5, 3.14, 2
> FcStart
> StartOverlay
> PauseOverlay
> RestartOverlay
> Restart the overlays.

Use restrictions

> ➢ The interface can only be called after executing PauseOverlay. This interface is used in conjunction with PauseOverlay to restart paused overlays.

12.3.4.18 StopOverlay

Explanation

> Stop the overlays.

Definition

> StopOverlay
> No parameters, and can be used directly.

Example

Example 1

> FcInit Tool1, Wobj0, 0
> SetControlType 1
> SetSineOverlay 0, 10, 5, 3.14, 2
> FcStart
> StartOverlay
> StopOverlay

Stop the overlays.

Use restrictions

> ➤ The calling of the interface is of practical value can only after executing StartOverlay.

12.3.4.19 FcCondForce

Explanation

It is used to define termination conditions related to contact force.

Definition

FcCondForce xmin, xmax, ymin, ymax, zmin, zmax, IsInside, TimeOut

xmin

Define the lower limit of the force limit in the X-direction. It indicates the maximum value in the negative X-direction if the value is negative. The unit is N and the default value is negative infinity. Data type: double

xmax

Define the upper limit of the force limit in the X-direction. It indicates the minimum value in the negative X direction if the value is negative. The unit is N and the default value is positive infinity. Data type: double

ymin

Define the lower limit of the force limit in the Y-direction. It indicates the maximum value in the negative Y direction if the value is negative. The unit is N and the default value is negative infinity. Data type: double

ymax

Define the upper limit of the force limit in the Y-direction. It indicates the minimum value in the negative Y direction if the value is negative. The unit is N and the default value is positive infinity. Data type: double

zmin

Define the lower limit of the force limit in the Z-direction. It indicates the maximum value in the negative Z direction if the value is negative. The unit is N and the default value is negative infinity. Data type: double

zmax

Define the upper limit of the force limit in the Z-direction. It indicates the minimum value in the negative Z direction if the value is negative. The unit is N and the default value is positive infinity. Data type: double

IsInside

It is used to define whether the internal/external restriction condition is true. Data type: bool

TimeOut

It is used to define the timeout period in seconds. The value taken ranges from 1 to 600. Data type: double

Example

Example 1

FcInit Tool1, Wobj0, 0
FcStart
FcCondForce -100, 100, -100, 100, -100, 100, true, 60
Define a termination condition. The condition is true when the contact force is within plus or minus 100N in the x/y/z-axis direction of the force control frame, and terminates when it exceeds 100N. The timeout period is 60 seconds.

Use restrictions

> ➤ This interface can only be called after executing FcStart and before executing FcStop. If not, the

termination conditions of the contact force will not be set successfully.

12.3.4.20 FcCondPosBox

Explanation	
	It is used to define termination conditions related to contact location.

Definition	
	FcCondPosBox SupvFrame, Box, IsInside, Timeout
SupvFrame	It is used to determine in which frame the monitored spatial body will be defined. The frame is derived by converting a work object frame onto a frame. The conversion of the frame is defined by pose. By default, pose0 is used. That is, the work object frame is used without using any conversion. Data type: pose
Box	Define a cuboid. Data type: fcboxvol
IsInside	It is used to define whether the internal/external restriction condition is true. Data type: bool
TimeOut	It is used to define the timeout period in seconds. The value taken ranges from 1 to 600. Data type: double

Example	
Example 1	FcInit Tool1, Wobj0, 0
	FcStart
	VAR fcboxvol box1 = fcbv:{-100.0, 100.0, -200.0, 200.0, -300.0, 300.0}
	VAR pose pose1 = pe:{0, 0, 0},{1, 0, 0, 0}
	FCCondPosBox pose1, box1, false, 60
	Define a termination condition. The termination condition is triggered when the robot TCP enters the defined cuboid or waits more than 60 seconds.

Use restrictions	
	➢ This interface can only be called after executing FcStart and before executing FcStop. If not, the termination conditions of the cuboid location will not be set successfully.

12.3.4.21 FcCondTorque

Explanation	
	It is used to define termination conditions related to contact torque.

Definition	
	FcCondTorque xmin, xmax, ymin, ymax, zmin, zmax, IsInside, TimeOut
xmin	Define the lower limit of the torque limit in the X-direction. It indicates the maximum value in the negative X-direction if the value is negative. The unit is N.m and the default value is negative infinity. Data type: double
xmax	

Define the upper limit of the torque limit in the X-direction. It indicates the minimum value in the negative X-direction if the value is negative. The unit is N.m and the default value is positive infinity.

Data type: double

ymin

Define the lower limit of the torque limit in the Y-direction. It indicates the maximum value in the negative Y-direction if the value is negative. The unit is N.m and the default value is negative infinity.

Data type: double

ymax

Define the upper limit of the torque limit in the Y-direction. It indicates the minimum value in the negative Y-direction if the value is negative. The unit is N.m and the default value is positive infinity.

Data type: double

zmin

Define the lower limit of the torque limit in the Z-direction. It indicates the maximum value in the negative Z-direction if the value is negative. The unit is N.m and the default value is negative infinity.

Data type: double

zmax

Define the upper limit of the torque limit in the Z-direction. It indicates the minimum value in the negative Z-direction if the value is negative. The unit is N.m and the default value is positive infinity.

Data type: double

IsInside

It is used to define whether the internal/external restriction condition is true.

Data type: bool

TimeOut

It is used to define the timeout period in seconds. The value taken ranges from 1 to 600.

Data type: double

Example

Example 1

FcInit Tool1, Wobj0, 0
FcStart
FcCondTorque -10, 10, -10, 10, -10, 10, true, 60

Define a termination condition. When the contact torque is greater than 10 Nm in any direction of the force control frame, or the time exceeds 60s, the termination condition is triggered.

Use restrictions

➢ This interface can only be called after executing FcStart and before executing FcStop. If not, the termination conditions of the contact torque will not be set successfully.

12.3.4.22 FcCondWaitWhile

Explanation

It is used to activate the previously defined termination conditions and wait until these conditions become False or timeout in the current line.

Definition

FcCondWaitWhile
No parameters, and can be used directly.

Example

Example 1

FcInit Tool1, Wobj0, 0
FcStart

FcCondTorque -10, 10, -10, 10, -10, 10, true, 60

FcCondForce -100, 100, -100, 100, -100, 100, true, 60

FcCondWaitWhile

Activates the termination conditions. The program blocks at the current position and waits for the termination conditions to be triggered.

Use restrictions

> It can be used after the force control termination conditions are defined.

12.3.4.23 GetEndToolTorque

Explanation

It is used to get the current robot torque

Definition

GetEndToolTorque Tool, Wobj [, RefType]

The parameter in [] can be ignored.

Return value

Torque information

Data type: TorqueInfo

Parameter

Tool

The information of the tool currently in use.

Data type: Tool

Wobj

The information of the work object currently in use.

Data type: Wobj

RefType

Reference frame relative to the torque

Data type: Int

0: Default. Torque information of the end-effector relative to the world frame

1: Torque information of the end-effector relative to the flange frame

2: Torque information of the end-effector relative to the TCP

Example

Example 1

FcInit Tool1, Wobj0, 0

FcStart

FcCondTorque -10, 10, -10, 10, -10, 10, true, 60

FcCondForce -100, 100, -100, 100, -100, 100, true, 60

FcCondWaitWhile

Activates the termination conditions. The program blocks at the current position and waits for the termination conditions to be triggered.

Use restrictions

> It can be used after the force control termination conditions are defined.

12.3.5 Drag and replay

12.3.5.1 ReplayPath

Explanation	
	Replays the recorded trajectory using drag teaching. You can control the running rate during replay. Refer to 4.2.4 Path Replay.

Definition	
	ReplayPath path [, rate] [, wobj/tool]
path	
	Data type: path
	Type of playback of recorded path which is defined in the variable list generated by drag teaching.
rate	
	Data type: double
	Replay percentage, 0.01-3.00. 0.01 means replay at 1% running rate when dragging; 1.00 at 100% running rate; 3.00 at 300% running rate.
wobj/tool	
	Data type: tool|work object
	Specify the end-effector for the replay command to be a tool or work object. During the replay, the robot will change the replay control parameters according to the tool of the corresponding device to improve the operating stability

Example	
Example 1	
	ReplayPath path , 1, tool1
	Use the original running rate to record and replay.

12.3.6 IO commands

12.3.6.1 SetDO

Explanation	
	It is used to set the value of a digital output signal.

Definition	
	SetDO DoName, Value
DoName	
	Data type: signaldo
	Specify the name of the DO signal whose state should be changed. It must be a variable that has already been defined on the IO interface.
Value	
	Data type: bool
	The target state of signaldo. Only true and false are supported.

Example	
Example 1	
	SetDO do2, true
	Set the digital output point corresponding to do2 as high level.

12.3.6.2 SetAllDO

Explanation	
	It is used to set the value of all digital output signals.

Definition

SetAllDO Value

Value

Data type: bool
The target state of signaldo. Only true and false are supported.

Example

Example 1

SetAllDO true
Set all digital output voltages to a high level, except DO bound with system function.

12.3.6.3 SetGO

Explanation

It is used to set the value output of a group.

Definition

SetGO GoName, Value

GoName

Data type: signalgo
Specify the name of the go signal whose value should be changed. It must be a variable that has already been defined on the IO interface.

Value

Data type: int
The target value of the go signal.

Example

Example 1

SetGO go3, 8
Set the value of a set of physical ports corresponding to go3 as 8.

12.3.6.4 SetAO

Explanation

It is used to set the value of an analog output signal.

Definition

SetAO AoName, Value

AoName

Data type: signalao
Specify the name of the ao signal whose value should be changed. It must be a variable that has already been defined on the IO interface.

Value

Data type: double
The target value of the ao signal.

Example

Example 1

SetAO ao3, 5.123
Set the value of a set of physical ports corresponding to ao3 as 5.123.

12.3.6.5 PulseDO

Explanation

To generate a pulse of a DO signal.

Definition	
	PulseDO [\High,] [length,] signal
[\High]	
(1).	When the command is executed, regardless of the current state, the signal state is always set to high
[length]	
	Specify pulse length: 0.001-2000s. Default to 0.2s when missing. Data type: double or int
signal	
	The signal to generate the pulse. Data type: signaldo

Use restrictions
➢ If SetDO/SetGO is executed during PulseDO, PulseDO will be invalid and SetDO/SetGO will be executed.

12.3.6.6 PulseReg

Explanation	
	Specify a register to generate a pulse signal for a specified time period and restore the initial value of the register after the time period ends.

Definition	
	PulseReg Register, Value, Time
Register	
	The name of the register to generate the pulse signal Data type: Bit|Bool register
Value	
	Specify the value of the pulse signal. Data type: Bool,
Time	
	The duration of the pulse signal in seconds, with a limit range of [0.001, 10.0]. Data type: double

Use restrictions
➢ If WriteRegByName or register equal assignment is executed during PulseReg, the valid value of the register will take effect depending on the last executed command. But the initial value before executing PulseReg will be restored after the time period specified by PulseReg ends.

12.3.7 Communication commands

In the RL program, the robot can communicate with external devices through both Ethernet and serial ports. A unified set of commands is designed for resource management and data sending and receiving, which ensures consistent use experience.

Command set	TCP client	TCP server	Serial port
OpenDev	√	√	√
SocketAccept	N/A	√	N/A
CloseDev	√	√	√
SendString	√	√	√
SendByte	√	√	√
ReadBit	√	√	√
ReadByte	√	√	√
ReadDouble	√	√	N/A
ReadInt	√	√	N/A

ReadString	√	√	√
GetSocketConn	√	N/A	N/A
GetSocketServer	N/A	√	N/A
GetBufSize	N/A	N/A	√
ClearBuffer	N/A	N/A	√

12.3.7.1 OpenDev

Explanation

Used to open a listening server, initiate a connection as a client, and open a serial port resource, depending on the object indicated by the parameter.

1) When opening the SocketServer object, the robot will initiate resource and complete port binding and port listening.

2) When opening the SocketConn object, the robot will act as a TCP client and try to connect to the external server according to the preset ip and port.

3) When opening the serial port resource, the serial port will be initialized according to the window parameters and communication conditions will be provided.

Return value

N/A.

Definition

OpenDev(name)

name

Data type: string
The name of the client object or server object or serial port resource.

Example

Example 1

```
SocketConn scnn3 = {"192.168.0.200", 8090, "clt1", 2, "\n"}
try
    OpenDev("clt1") // Try to connect to the remote server. If the connection is successful, the attr
of clt1 will be modified to outgoing automatically.
    string readstr = ReadString(30, "clt1")
    .....    // Logic processing of readstr
    string sendstr = "hello server!"
    SendString(sendstr , "clt1") //Use clt1's client connection to send data
    ...      // A series of code
    catch(ERROR e) // ERROR error type, including the file that generated the error, line number,
error code, and error content
    ...      // A series of exception handling
Endtry
```

Example 2

```
SocketServer listener1  = {"192.168.0.200", 8090, "svr1"}
global pers bool exit = false
try
    OpenDev( "svr1" ) // Bind port, listen for port
    while(exit != true)
        SocketConn conn = SocketAccept( "svr1") // Client connected via blocking receive
    Endwhile
catch(ERROR e)
...      // A series of exception handling
Endtry
```

Error handling

If an error is reported, the control system will throw an exception and report the cause of the error.

If the exception is not caught by the try block, the control system will stop the program.

12.3.7.2 SocketAccept

Explanation

Blocking wait for client connections to arrive, and complete client connection. This command is only used when the robot is acting as a TCP server.

> **Notes**
>
> 1. The command will block the current task, so the correct way to use it is in multitasking. There is a low-priority task continuously receiving and generating the communication connection object SocketConn independently.
> 2. The command returns a connection operation object and has the ip and port information of the client connection, which can be used by other parts of the program. The returned connection object is a SocketConn structure with a name randomly assigned by the system. After getting the connection object, please change the name of the connection object to avoid connection loss.
> 3. The server supports multiple connections.

Return value

Data type: SocketConn

After an external device connects to the robot as a TCP client, the control system generates a communication object that is used by the RL program to control communication read and write.

Definition

SocketConn conn = SocketAccept(name)

name

Data type: string

The name of the SocketServer object that has been prepared and opened successfully using OpenDev.

Examples

Example 1

```
SocketServer listener1 = {"192.168.0.200", 8090, "svr1"}
global pers bool exit = false
try
    OpenDev( "svr1" ) // Bind port, listen for port
    while(exit != true)
        SocketConn conn = SocketAccept( "svr1") // Client connected via blocking receive
        conn.name = "client1" // Important! Give the communication connection a name, otherwise, it
        will be difficult to read and write data by name
        conn.suffix = "\n" // Optional, set the packet terminator
    Endwhile
catch(ERROR e)
...      // A series of exception handling
Endtry
```

Error handling

If an error is reported, the control system will throw an exception and report the cause of the error. If the exception is not caught by the try block, the control system will stop the program.

12.3.7.3 CloseDev

Explanation

Close the resource, which can be used to close the TCP communication connection, TCP listening server, or serial port resource.

Return value

N/A.

Definition

CloseDev(name)

name

Data type: string

SocketConn connection, listening server SocketServer object, or serial port resource used for communication.

Examples

Example 1

```
SocketConn scnn3 = {"192.168.0.200", 8090, "clt1", 2, "\n"}
try
    OpenDev("clt1")
    string readstr = ReadString(30, "clt1")
    .....     // Logic processing of readstr
    string sendstr = "hello server!"
    SendString (sendstr , "clt1") //Use clt1's client connection to send data
    ...       // A series of code
catch(ERROR e)
    ...       // A series of exception handling
endtry
CloseDev("clt") // Close the socket client at last, regardless of whether an error occurs.
```

Example 2

```
SocketServer listener1  = {"192.168.0.200", 8090, "svr1"}
global pers bool exit = false
try
    OpenDev( "svr1" ) // Bind port, listen for port
    while(exit != true)
        SocketConn conn = SocketAccept( "svr1") // Client connected via blocking receive
        conn.name = "client1" // Important! Give the communication connection a name, otherwise, it
        will be difficult to read and write data by name
        conn.suffix = "\n" // Optional, set the packet terminator
    Endwhile
catch(ERROR e)
...       // A series of exception handling
Endtry

CloseDev("client1") // Close communication with external TCP client. Important!
CloseDev("svr1") // Close the listening server
```

> **ℹ Notes**
>
> 1. In Example 2, there are two network objects, and you must close the communication connection first and then the server object, otherwise it will generate a state of incomplete resource release (TCP TIME_WAIT state).
> 2. If the robot has established multiple communication connections with external devices when it acts as a server, you need to close these communication connections in order before closing the server.
> 3. In the case of incomplete resource release, the control system needs to be restarted. However, there is no need to worry too much, as there is redundancy in the number of resources allowed in the control system; this ensures the program runs properly after a small number of resources are occupied. However, it is necessary to avoid a large number of resources being occupied due to incorrect use.

12.3.7.4 SendString

Explanation

Send a string outwards. It can be sent through the network or serial port, depending on the hardware resource represented by the identifier in the parameter.

Definition

	SendString(StringData, name)
StringData	
	Data type: string
	The string data to be sent.
name	
	Data type: string
	The name of the hardware resource used to send the data. It can be the SocketConn object with an established TCP communication connection or the serial port resource successfully opened.

Example	
Example 1	
	SendString("Hello World", "Socket0")
	Send Hello World string outwards through Socket0. Socket0 is the SocketConn type that has been defined and successfully connected.
Example 2	
	VAR String str1 = "Hello World"
	SocketSendString(str1, "Serial1")
	Sends the string Hello World stored in str1 outwards via Serial1. Serial1 is a defined and successfully opened serial port.

12.3.7.5 SendByte

Explanation	
	To send a byte outwards. It is very useful when sending ASCII characters.

Return value	
	N/A.

Definition	
	SendByte(ByteData, name)
ByteData	
	Data type: int, byte, or byte array
	Send an unsigned byte or array from 0 to 255, mainly used for sending ASCII codes.
name	
	Data type: string
	The name of the socket or serial port to send data.

Example	
Example 1	
	SendByte(13, "socket0")
	Send a carriage return through Socket0.
Example 2	
	VAR byte data1 = 13
	SendByte(data1, "serial0")
	First define a byte variable data1, which is actually a carriage return. Then send the data outwards through serial0.
Example 3	
	VAR byte data2[2] = {13,17}
	SendByte(data2, "socket0")
	Send an array variable byte data2 through socket0. Sent all in the array.
Example 4	
	VAR byte data2[2] = {13,17,20}
	SendByte(data2[2], "socket0")
	Sends a byte variable of data2[2] through socket0, which represents the 2nd element of the array.The value 17 of data2[2] will be sent without sending any other elements.

12.3.7.6 ReadBit

Explanation	
	The control system receives data by bit.

1) Received by TCP through network communication. The externally sent data should end with the terminator configured by SocketConn.

2) Received by serial communication. The external device only needs to send the data, with no requirement on the terminator.

Return value	
	Data type: bool array Store the received bit data using a bool array. Each bit corresponds to a bool member.

Definition	
	Ret = ReadBit(BitNum, TimeOut, name)
BitNum	Data type: int The number of bits that need to be read. The size should be an integer multiple of 8.
TimeOut	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
name	Data type: string The name of the communication connection SocketConn or the serial port.
Ret	Data type: bool array Received data. The first element of the array indicates the lowest bit.

Example	
Example 1	
	bool groupio[16] groupio = ReadBit(16, 60, "Socket0") 16 bit data is read by the SocketReadBit command and stored in a bool array named groupio with a timeout period of 60 seconds. Assume that the external device sends ASCII characters, 95 + terminator, the robot receives "95". As the hexadecimal values of "9" and "5" are 0x39 and 0x35 respectively, the data received by the user is 0x3935. At this time the groupio array from [1] to [16] is 1001 1100 1010 1100. The [1] is the low bit of the data, which matches with 0x3935.

12.3.7.7 ReadByte

Explanation	
	Receive data with a certain number of bytes. Note that the data needs to be separated by commas.

Return value	
	Data type: byte array Store the received data using a byte array.

Definition	
	Ret = ReadByte(ByteNum, TimeOut, name)
ByteNum	Data type: int The number of bits that need to be read. The size should be an integer multiple of 8.
TimeOut	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
name	Data type: string The name of the communication connection SocketConn or the serial port.
Ret	Data type: byte array Received data.

Example	

Example 1

> byte rets[6] = {0,0,0,0,0,0}
> rets = ReadByte(6,60,"clt1")
> 6-byte data is read and stored in a bool array named rets with a timeout period of 60 seconds.
> Note that bytes from external devices need to be separated by commas, e.g. send "1,2,3,4,5,6"
> When sending data via TCP, the data should end with the pre-defined terminator.
> When sending data via serial port, the terminator is not required.

12.3.7.8 ReadDouble

Explanation	
terminator.	It is used to receive double-type data via Socket.The sent data should end with the pre-defined
	Note that this command is only valid for TCP network communication and when robots act as the client/server, but not for serial ports.

Return value	
	Data type: double array Store the received data using a double array.

Definition	
	Ret = ReadDouble(DoubleNum, TimeOut, name)
DoubleNum	
	Data type: double The number of doubles to be read, up to 30.
TimeOut	
	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
name	
	Data type: string The name of the Socket used to receive the data.

Example	
Example 1	
	double dd[10] dd =ReadDouble(10, 60, "Socket0") Read 10 double-type data and store them in a double array named dd with a timeout period of 60 seconds.

12.3.7.9 ReadInt

Explanation	
	It is used to receive int-type data via Socket. Externally sent data must end with the pre-defined terminator. Note that this command is only valid for TCP network communication and when robots act as the client/server, but not for serial ports.

Return value	
	Data type: int Store the received data using an int array.

Definition	
	Ret = ReadInt(IntNum, TimeOut, name)
IntNum	
	Data type: int The number of int to be read, up to 30.
TimeOut	
	Data type: int

name	Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
	Data type: string The name of the Socket used to receive the data.

Example	
Example 1	
	int ii[10] ii = ReadInt(10, 60, "Socket0") 10 int data are read and stored in an int array named ii with a timeout period of 60 seconds.

12.3.7.10 ReadString

Explanation	
	It is used to read a string and return it. Externally sent data should end with the pre-defined
terminator.	

Return value	
	Data type: string Store the received string.

Definition	
	Ret = ReadString(TimeOut, name, [len])
TimeOut	
	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
name	
	Data type: string The name of the socket or serial port to receive data.
len	
	Data type: int Optional parameter, only used when reading through the serial port. Since the terminator is not defined in the serial port, it is necessary to specify the length before successful reading and parsing.

Example	
Example 1	
	VAR String str1 str1 = ReadString(60, "Socket1") Receive a string from Socket1 and store it in str1 with a timeout period of 60 seconds. Network Communication
Example 2	
	VAR String str1 str1 = ReadString(60, "serial0",5) Receive a string for a length of 5 bytes from serial0 and store it in str1 with a timeout period of 60 seconds. Serial port communication method.

12.3.7.11 GetSocketConn

Explanation	
	It is used to find the socket attribute set object using the socket connection name. The result obtained by this command can be used for judgment and processing logic. It should be used only as a read-only object. This command is only applicable to communication connections (including robot as client, or as a server which has been connected to the channel for communication), not for listening servers and serial ports.

Return value	
	Data type: SocketConn The socket attribute object found by given name.

Definition

Ret = GetSocketConn(name)

name

Data type: string
Name of communication connection SocketConn.

Ret

Data type: SocketConn
The socket attribute object found by given name.

Example

Example 1

SocketConn ret= GetSocketConn("client0")
Find SocketConn object with the name "client0". You can use ret to get the attributes of this
connection, including the ip address, port number, communication terminator, and connection state.

Queryable properties	Query method	Meaning and example
ip address	ret.ip	String, e.g. "192.168.0.161"
Port number	ret.port	integer, e.g. 8090
Attribute	ret.attr	Robot as server: "incoming". Robot as client: "outgoing". If the connection is not established: "" or other value, usually blank
Cache size	ret.cache	1~100
Name	ret.name	In the given example, it is "client0"
Connection state	ret.state	closed, established

12.3.7.12 GetSocketServer

Explanation

Find the corresponding server attribute set object with the user-defined name. The result obtained by
this command can be used for judgment and processing logic. It should be used only as a read-only
object. This command is only applicable to listening servers (SocketServer objects), not to
communication connections (including robot as client, or as a server which has been connected to the
channel for communication) and serial ports.

Return value

Data type: SocketServer
The server attribute object found by given name.

Definition

Ret = GetSocketServer(name)

name

Data type: string
Name of communication connection SocketServer.

Ret

Data type: SocketServer
The socket attribute object found by given name.

Example

Example 1

SocketServer listener1 = {"192.168.0.200", 8090, "svr1"}
OpenDev("svr1") // Bind port, listen for port
// Get the SocketServer object using the connection identifier "svr1", at this time ret will copy all the states of
listener1 in Task 1
SocketServer ret= GetSocketServer("svr1")
if(svrfindout.state == "listening") // Use SocketServer's attr attribute to judge if listening is in
underway
   // Logic processing
endif

Queryable properties	Query method	Meaning and example
ip address	ret.ip	String, e.g. "192.168.0.161"
Port number	ret.port	integer, e.g. 8090
Name	ret.name	In the example above, it is "svr1"
Connection state	ret.state	closed, listening, error

12.3.7.13 GetBufSize

Explanation	
	Get the amount of data not read in the buffer of the serial port, in bytes. The command is only applicable to the serial port, not to the TCP server and the client.

Return value	
	Data type: int The amount of unprocessed data in the buffer, in bytes.

Definition

Ret = GetBufSize(name)

name

Data type: string
The name of the serial port resource.

Ret

Data type: int
The amount of unprocessed data in the buffer, in bytes.

Example

Example 1

```
OpenDev("serial0")
int a = GetBufSize("serial0")
print(a)
```

12.3.7.14 ClearBuffer

Explanation	
	Clear the buffer. Any unread characters will be lost. The command is only applicable to the serial port, not to the TCP server and the client.

Return value	
	N/A.

Definition

ClearBuffer(name)

name

Data type: string
The name of the serial port resource.

Ret

Data type: int
The amount of unprocessed data in the buffer, in bytes.

Example

Example 1

```
OpenDev("serial0")
int a = GetBufSize("serial0")
print(a)
```

12.3.8 Network command

12.3.8.1 SocketCreate (expired)

Explanation	
	Establish a Socket connection. By using the Socket command, the RL program can obtain data from an external device or send out program data. The RL language supports the simultaneous establishment of multiple different Sockets for connections of multiple external devices. Different names should be used to distinguish between the different Sockets. The Socket command is based on the TCP/IP protocol, so theoretically any external device that supports TCP/IP can communicate with the RL program to exchange data. All data sent to the RL Socket command (i.e. data received using the SocketRead series of commands) should end with a carriage return. All data before the receipt of the carriage return will be merged into the same data processing. When using the Socket function, the robot controller only supports connection to an external server as a client.
	Up to 10 Socket connections are supported.
	Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.

Return value	
	Data type: bool
	Return true if created successfully and false if failed

Definition	
	SocketCreate("ip_Address", Port, "Name" [,Cache] [, "Terminator"])
ip_Address	
	Data type: string Define the IPv4 address that needs to be connected to the server. The double quotation marks shall be used to include it.
Port	
	Data type: int
	Define the server port number.
Name	
	Data type: string
	Define the name of a new Socket. Different names must be specified between different Sockets.
Cache	
	Data type: int
	Define the size of the Socket cache. The communication data is stored in the cache queue and can be omitted.
Terminator	
	Data type: string
	Define the terminator type of socket communication, which can be omitted, default to "\r".

Example	
Example 1	

```
if (SocketCreate("10.0.6.11",8080,"S1",10,"\r"))
   // Successful creation
else
   // Error handling
endif
```

> **ℹ Notes**
>
> 3、 Due to the limitation of the TCP/IP protocol resource release mechanism, do not call the commands SocketCreate and SocketClose frequently. Otherwise, the program may run incorrectly.

> 4、 To avoid frequent calls to the SocketCreate and SocketClose commands in loop mode, it
> is best to add a time delay between the two commands, e.g.
> SocketClose("S1")
> wait 0.1
> SocketCreate("10.0.6.11",8080,"S1",10,"\r")

12.3.8.2 SocketClose (expired)

Explanation

It is used to close the Socket.
Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It
is still valid in higher versions, but no longer maintained. Further use is not recommended.

Definition

SocketClose ("SocketName")

SocketName

Data type: string
The name of Socket to be closed.

> **i** Notes
>
> Do not use the SocketClose command directly after the SocketSend series of commands. Failure
> to do so may result in data transmission failures. Use the SocketClose command after receiving
> the confirmation messages.

Example

Example 1

SocketClose("Socket0")

12.3.8.3 SocketSendString (expired)

Explanation

It is used to send a string outwards via Socket.
Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It
is still valid in higher versions, but no longer maintained. Further use is not recommended.

Definition

SocketSendString(StringData, "SocketName")

StringData

Data type: string
The string data to be sent.

SocketName

Data type: string
The name of the Socket used to send the data.

Example

Example 1

SocketSendString ("Hello World", "Socket0")
Send Hello World string outwards through Socket0.

Example 2

VAR String str1 = "Hello World"
SocketSendString(str1, "Socket0")
Send the str1 stored string via Socket0.

12.3.8.4 SocketSendByte (expired)

Explanation	
	It is used to send a byte outwards through the Socket. It is very useful when sending ASCII characters. Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.
Definition	
	SocketSendByte(ByteData, "SocketName")
ByteData	Data type: int, byte, or byte array Send an unsigned byte or array from 0 to 255, mainly used for sending ASCII codes.
SocketName	Data type: string The name of the Socket used to send the data.
Example	
Example 1	SocketSendByte(13, "socket0") Send a carriage return through Socket0.
Example 2	VAR byte data1 = 13 SocketSendByte(data1, "socket0") First define a byte variable data1, which is actually a carriage return. Then send it outwards through Socket0.
Example 3	VAR byte data2[2] = {13,17} SocketSendByte(data2, "socket0") Send an array variable byte data2 through socket0.

12.3.8.5 SocketReadBit (expired)

Explanation	
	It is used to receive data by Bit through the Socket. Externally sent data must end with a carriage return. Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.
Return value	
	Data type: bool Store the received bit data using a bool array. Each bit corresponds to a bool member.
Definition	
	SocketReadBit(BitNum, TimeOut, "SocketName")
BitNum	Data type: int The number of bits that need to be read. The size should be an integer multiple of 8.
TimeOut	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
SocketName	Data type: string The name of the Socket used to receive the data.
Example	
Example 1	bool groupio[16] groupio = SocketReadBit(16, 60, "Socket0")

16 bit data is read by the SocketReadBit command and stored in a bool array named groupio with a timeout period of 60 seconds.

12.3.8.6 SocketReadDouble (expired)

Explanation	
	It is used to receive double-type data via Socket. Externally sent data must end with a carriage return. Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.

Return value	
	Data type: double Store the received data using a double array.

Definition	
	SocketReadDouble(DoubleNum, TimeOut, "SocketName")
DoubleNum	Data type: double The number of doubles to be read, up to 30.
TimeOut	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
SocketName	Data type: string The name of the Socket used to receive the data.

Example	
Example 1	
	double dd[10] dd = SocketReadDouble(10, 60, "Socket0") Read 10 double-type data using the SocketReadDouble command and store it in a double array named dd with a timeout period of 60 seconds.

12.3.8.7 SocketReadInt (expired)

Explanation	
	It is used to receive int-type data via Socket. Externally sent data must end with a carriage return. Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.

Return value	
	Data type: int Store the received data using an int array.

Definition	
	SocketReadInt(IntNum, TimeOut, "SocketName")
IntNum	Data type: int The number of int to be read, up to 30.
TimeOut	Data type: int Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
SocketName	Data type: string The name of the Socket used to receive the data.

Example	
Example 1	

int ii[10]
ii = SocketReadInt(10, 60, "Socket0")
10 int data is read by the SocketReadInt command and stored in an int array named ii with a timeout period of 60 seconds.

12.3.8.8 SocketReadString (expired)

Explanation	
	It is used to read a string from Socket and return it. Externally sent data should end with a carriage
return.	
	Note that this command is marked as "expired" for it is used in xCore Control System version 1.3. It is still valid in higher versions, but no longer maintained. Further use is not recommended.
Return value	
	Data type: string
	Store the received string.
Definition	
	SocketReadString(TimeOut, "SocketName")
TimeOut	
	Data type: int
	Timeout period, in s, ranging from 0 to 86400 and is defaulted to 60s.
SocketName	
	Data type: string
	The name of the Socket used to receive the data.
Example	
Example 1	
	VAR String str1
	str1 = SocketReadString(60, Socket1)
	Receive a string from Socket1 and store it in str1 with a timeout period of 60 seconds.

12.3.9 Logic commands

12.3.9.1 Return

Explanation	
	Function or TRAP return.
	When the program encounters a RETURN command, if the program is currently in a subroutine or TRAP, the program will return to the previous function. If the program is currently in the main function, the program ends directly.

12.3.9.2 Wait

Explanation	
	The program waits for a period of time ranging from 0 to 2147484 seconds.
Example	
Example 1	
	Wait 2
	Indicates waiting for 2 seconds.

12.3.9.3 WaitUntil

Explanation	
	The program waits until a certain condition is met.

Example	
Example 1	
	WaitUntil(di2 == true)
	It indicates that the second signal waiting for the second signal of the first DI module is true before executing sentences followed.

12.3.9.4 Break

Explanation	
	Jumping out of the current loop, and is used in the WHILE loop in the RL language. When the WHILE loop is executed to Break, regardless of WHILE's CONDITION, it will jump out from the WHILE loop directly.

Example	
Example 1	

```
VAR int counter = 0
WHILE(1)
  IF(counter == 5)
      break
  Endif
counter++
ENDWHILE
```
The program will jump out of the WHILE loop when the counter is 5.

12.3.9.5 IF…Else if…Else

Explanation	
	Conditional judgment command.

Example	
Example 1	

```
IF(condition1)
    //a
Else if (condition2)
    //b
Else if (condition3)
    //c
Else
//d
Endif
```
Execute logic a when condition1 is true, logic b when condition2 is true, and so on.

12.3.9.6 Goto

Explanation

The Goto command allows the pointer to jump to the marked command.

Example

Example 1

> int a = 0
> int b = 9
> Goto end
> printf(a)

end:

> printf(b)

Define two variables a and b, then use the printf function to print two commands. Use the Goto command to force a jump to the end marker position of the print b command, at which point the print of a will not be executed.

12.3.9.7 For

Explanation

Defines a loop control structure that executes a specified number of times.

Example

Example 1

> For(int i from 1 to 10)
>   printf("i = %d\n", i)
> endfor

This program prints i 10 times from 1 to 10 by adding 1 each time in sequence.

Example 2

> For(int i from 1 to 10 step 3)
>   printf("i = %d\n", i)
> Endfor

This program prints i 4 times from 1 to 10 by adding 3 each time in sequence.

Supplementary explanation

Continue and Break can be used to control the For flow. See the Continue and Break commands for details.

12.3.9.8 Continue

Explanation

Exit this loop.

Continue executing the commands from the beginning of the loop, but just end the loop without exiting from the loop body.

Example

Example 1

> VAR int count = 0
> WHILE(1)

```
    count++
    IF(count == 1)
      Continue
    Else
      break
      MoveAbsJ j10, v500, fine, tool1
    Endif
ENDWHILE.
The code for MoveAbsJ will not be executed.
```

12.3.9.9 Inzone

Explanation

It is used with SetDO or modbus, cclink, and other IO operations or commands; this command can ensure that the signal is triggered at a defined point position, instead of being triggered earlier by the lookahead pointer.

Example

```
MoveL p1
MoveL p2
Inzone
    SetDO dox, true
    print(123)
EndInzone
MoveL p3
```

Supplementary explanation

In the example, an Inzone command is used. After the interpreter looks ahead to Inzone, instead of executing this command immediately, it generates an additional function which includes SetDo and print commands. This additional function takes effect when the motion command move p2 is completed.
1. If there is a turning zone between the two motion commands p2 and p3, the additional function will be executed at the moment when the robot reaches the turning zone
2. If there is no turning zone, the additional function will be executed at the moment the robot reaches p2

12.3.9.10 WHILE

Explanation

While loop allows you to write a loop control structure that keeps executing before conditions are met.

Example

Example 1

```
int count = 0
while(count < 10)
    count++
    print(count)
endwhile
```
This program enables a loop that counts by 1 from 0 to 10 and prints.

Supplementary explanation

Continue and Break can be used to control the While flow. See the Continue and Break commands for details.

12.3.9.11 Pause

Explanation

It is used to pause the program.

The program enters the pause state after the command before Pause is executed. The program can only be resumed by clicking running on the teach pendant or receiving a restart signal through an external program.

> **ℹ Notes**
>
> This command does not support auxiliary programming for the moment.

12.3.9.12 try/catch

Explanation

The Try/Catch command allows the program writer to decide unique response measures after an error occurs.

Example

Example 1

```
ReadOnce:
Try
    Double xyz[3] = ReadDouble(3, timeout, socketname)
    Robtarget_0.trans.x = xyz[1]
    Robtarget_0.trans.y = xyz[2]
    Robtarget_0.trans.z = xyz[3]

    MoveL Robtarget_0, v2000, fine, tool0
Catch(error e)
    SendString("Recv rob xyz error", socketname)
    Goto ReadOnce
endtry
```

This program supports a simple application scenario. The communication command ReadDouble is used to read a three-dimensional array from the TcpSocket as the xyz parameter of the motion point position, and then the MoveL command is called to move to the corresponding Cartesian point.

If the try/catch command is not used and the point position received from the TcpSocket is wrong, the robot will report "out of range" or "planning error" and stop the program.

If the try/catch command is used, the motion command error is still reported, but the program does not stop. Instead, it jumps to the code segment between catch and endtry and handles the error as desired by the user. In this example, SendString tells Socket the point position error received by the host, and the host decides how to handle the error and calls the goto command to re-execute ReadDouble and wait for the next position.

Commands whose errors can be caught by the try/catch command:

All motion commands (see 12.3.2 Motion commands)

All communication commands (see 12.3.7 Communication commands)

12.3.9.13 SwitchCase

Explanation

SwitchCase, like the IF command, controls the flow control based on the input variable conditions. RL interpreter will compare the variables in the Case field in order based on the input variable (condition).

If the two variables are equal, the interpreter will enter the code branch of the corresponding Case and stop comparing and entering other code branches.

If all conditions are not met, it will enter the Default branch;

If no Case condition matches and there is no Default branch, it will enter no branch and the Switch command ends;

Multiple conditions can be input for the Case command (see command structure Case C1, C12, C13 and example 1).

Command structure

```
Switch(condition)
    Case C1,C12,C13:
        Functions1()
    Case C2:
        Functions2()
    Default:
        DefaultFunction()
EndSwitch
```

Example

Example 1

reg_int is a register variable, the host (PLC) will update the value of the variable through relevant register protocols (e.g. modbus, cclink). The production project expects the robot to execute the corresponding function branch (e.g. a blocked trajectory) according to the value of the register. If the register inputs 1, 2, and 3, then function A will be executed; if the register inputs 4, 5, and 6, then function B is executed. If the above conditions are not met, function C will be executed in the Default branch.

```
Switch(reg_int)
    Case 1,2,3:
        FunctionsA()    // The robot follows point positions related to function A
    Case 4,5,6:
        FunctionsB()    // The robot follows point positions related to function B
    Default:
        FunctionC()     // Execute function C if without specified input
EndSwitch
```

12.3.10 Home command

12.3.10.1 Home

Explanation

Make the robot return to the set Home through joint space motion.

Definition	
	Home
	The command includes no input parameters

Example	
Example 1	
	HomeSet 0,30,0,60,0,90,0
	Home
	Use the HomeSet command to set the Home and then the Home command to move the robot to the drag pose in the joint space.

Use restrictions	
	➢ Home pose setting must be enabled on the Robot Setup > Quick Turn interface or through the HomeSet command before the Home command can be used, otherwise, an error is reported.

12.3.10.2 HomeSet

Explanation	
	Sets the robot's Home in the joint space

Definition	
	HomeSet axis1,axis2,axis3,axis4,axis5,axis6,axis7
axisx	
	Data type: double
	Set the angle of home on each axis

Example	
Example 1	
	HomeSet 0,30,0,60,0,90,0
	Home
	Use the HomeSet command to set the Home and then the Home command to move the robot to the drag pose in the joint space.

12.3.10.3 HomeSetAt

Explanation	
	Obtain the setup data of the robot's Home

Definition	
	HomeSetAt(index)
Return value	
	Data type: double
	Joint angle, in °
index	
	Data type: int
	Get the joint angle of the specified axis at Home. When the index is 0, return if HomeSet is enabled, 1 means enabled, and 0 disabled.

Example	
Example 1	

HomeSet 0,30,0,60,0,90,0

double angle2 = HomeSetAt(2)

angle2 Get the joint angle of joint 2 at 30°.

12.3.10.4 HomeDef

Explanation	
	Determine if the Home is set
Definition	
	HomeDef()
Return value	
	Data type: bool
	true Home already set
	false Home not set

12.3.10.5 HomeSpeed

Explanation	
	Set the running speed of Home command
Definition	
	HomeSpeed Speed
Example	
Example 1	
	HomeSpeed v1000
	Home
	Set the Home speed to V1000. Then the Home command moves the robot to Home at the speed of V1000.

12.3.10.6 HomeClr

Explanation	
	Clear Home setting
Definition	
	HomeClr
Example	
Example 1	
	HomeClr
	Clear Home set in the program. The Home command will not be executed if cleared.

12.3.11 Math command

12.3.11.1 sin

Function definition: double sin(double x);
Description: sin() is used to calculate the sine of parameter x and return the result. x in radians;
Return value: Return the calculated result between -1 and 1.

12.3.11.2 cos

Function definition: double cos(double x);
Description: cos() is used to calculate the cosine of parameter x and return the result. x in radians;
Return value: Return the calculated result between -1 and 1.

12.3.11.3 tan

Function definition: double tan(double x);
Description: tan() is used to calculate the tangent of parameter x and return the result. x in radians;
Return value: Return the tangent of parameter x.

12.3.11.4 cot

Function definition: double cot(double x);
Description: cot() is used to calculate the cotangent of parameter x and return the result. x in radians;
Return value: Return the cotangent of the parameter x.

12.3.11.5 asin

Function definition: double asin(double x);
Description: asin() is used to calculate the arcsine of parameter x and return the result. Parameter x ranges from -1 to 1,
beyond which error will be reported.
Return value: Return the calculated result between -PI/2 and PI/2, in radians.

12.3.11.6 acos

Function definition: double acos(double x);
Description: acos() is used to calculate the arccosine of parameter x and return the result. Parameter x ranges from -1 to 1,
beyond which error will be reported.
Return value: Return the calculated result between 0 and PI, in radians.

12.3.11.7 atan

Function definition: double atan(double x);

Description: atan() is used to calculate the arctangent value of parameter x and return the result.

Return value: Return the calculated result between -PI/2 and PI/2.

12.3.11.8 sinh

Function definition: double sinh(double x)

Description: sinh() is used to calculate the hyperbolic sine value of parameter x and return the result.

The mathematical definition is:

(exp(x) - exp(-x))/2;

Return value: Return the hyperbolic sine of parameter x.

12.3.11.9 cosh

Function definition: double cosh(double x)

Description: cosh() is used to calculate the hyperbolic cosine of parameter x and return the result.

The mathematical definition is:

(exp(x)+exp(x))/2;

Return value: Return the hyperbolic cosine of the parameter x.

12.3.11.10 tanh

Function definition: double tanh(double x);

Description: tanh() is used to calculate the hyperbolic tangent of parameter x and return the result.

The mathematical definition is:

sinh(x)/cosh(x);

Return value: Return the hyperbolic tangent of parameter x.

12.3.11.11 exp

Function definition: double exp(double x);

Description: exp() is used to calculate e to the x power, which is the ex value, and return the result;

Return value: Return the result of e to the x power.

12.3.11.12 log

Function definition: double log(double x);

Description: log() is used to calculate the logarithm value of x at the base of e and return the result.

That is, to find the natural logarithm of x,

ln(x)，x > 0；

Return value: Return the natural logarithm value of parameter x.

12.3.11.13 log10

Function definition: double log10(double x);
Description: log10() is used to calculate the logarithm value of x at the base of 10, and return the result. Where x>0;
Return value: Return the natural logarithm value of parameter x at the base of 10.

12.3.11.14 pow

Function definition: double pow(double x, double y);
Description: pow() is used to calculate x to the y power, which is the xy value, and return the result;
Return value: Return the result of x to the y power.

12.3.11.15 sqrt

Function definition: double sqrt(double x);
Description: sqrt() is used to calculate the square root of parameter x and return the result. The parameter x must be positive;
Return value: Return the square root of parameter x.

12.3.11.16 ceil

Function definition: double ceil(double x);
Description: ceil() will return the minimum integer value no less than parameter x, and the result will be returned in the double type.
Return value: Return a minimum integer value not less than the parameter x.

12.3.11.17 floor

Function definition: double floor(double x);
Description: floor() will return the maximum integer value not greater than the parameter x, and the result will be returned in the double type.
Return value: Return the maximum integer value not greater than the parameter x.

12.3.11.18 abs

Function definition: int abs(int x)/double abs(double x);
Description: Find the absolute value of x, |x|;
Return value: When the input parameter is of int type, the output is also of int type. When the input parameter is of double type, the output is also
of double type.

12.3.11.19 rand

Function definition: rand()

Function description: To generate an integer random number;

Return value: An integer random number, ranging from 0 to 2147483647.

12.3.12 Bit operation

12.3.12.1 BitAnd

Explanation

BitAnd is used to generate logical conjunction (and) for byte type data. See table below:
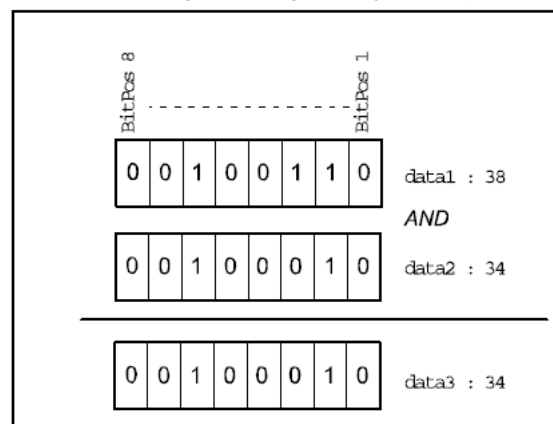
Return value

Data type: byte

It indicates the result returned by performing logical conjunction of two byte-type data.

Definition

BitAnd (BitData1, BitData2)

BitData1

Data type: byte

The byte data 1 to be processed.

BitData2

Data type: byte

The byte data 2 to be processed.

Example

Example 1

VAR byte data1 = 34

VAR byte data2 = 38

VAR byte byte3 = BitAnd(data1, data2) //34

Define the byte-type variable data1 and data2. assign them with the value of 34 and 38, respectively;

perform logical conjunction on data1 and data2, the returned value of 34 is assigned to byte3.

12.3.12.2 BitCheck

Explanation	
	To check whether a bit in a byte-type data is 1. If so, returns true, otherwise, false.
Return value	
	Data type: bool
	true indicates the bit is assigned to 1, false indicates the bit is assigned to 0.
Definition	
	BitCheck (BitData, BitPos)
BitData	
	Data type: byte
	Byte data to be operated.
BitPos	
	Data type: int
	Position of byte to be operated, ranging from 1 to 8.
Example	
Example 1	
	VAR byte data1 = 130
	VAR bool b1 = BitCheck(data1, 8) //true
	Definite byte data1 and assign it with 130, check if the 8th bit of data1 is 1 and return true if so.

12.3.12.3 BitClear

Explanation	
	To set a certain bit of byte- or int-type data to 0. The bit starts from 1.
Definition	
	BitClear BitData | IntData, BitPos
BitData	
	Data type: byte
	Byte data to be operated.
IntData	
	Data type: int
	The integer data to be operated.
BitPos	
	Data type: int
	Position of the bit to be operated, ranging from 1 to 8 for byte data and 1 to 32 for int data.
Example	
Example 1	
	VAR byte data1 = 255
	BitClear data1 1 //254
	BitClear data1 2 //252
	Define byte-type variable data1 and assign it with 255, Perform BitClear on data1, set the first bit to 0, and 254 is returned, set the second bit to 0, and 252 is returned.

12.3.12.4 BitLSh

Explanation	
	It is used to perform logical left shift on byte-type data.

Return value	
	Data type: byte
	It represents the byte data obtained by performing the left-shift operation.

Definition	
	BitLSh (BitData, ShiftSteps)
BitData	
	Data type: byte
	Byte data to be operated.
ShiftSteps	
	Data type: int
	The bits selected for the left shift, ranging from 1 to 8.

Example	
Example 1	
	VAR int left_shift = 3
	VAR byte data1 = 38
	VAR byte data2
	data2 = BiLSh(data1, left_shift) //48
	Define byte-type variable data1, and assign it with 38, perform 3 bits left shift on data1, and 48 is returned.

12.3.12.5 BitNeg

Explanation	
	It is used to perform logical negation on byte-type data.

Return value	
	Data type: byte
	It represents the byte data obtained by performing the logical negation.

Definition	
	BitNeg (BitData)
BitData	
	Data type: byte
	Byte data to be operated.

Example	
Example 1	
	VAR byte data1 = 38
	VAR byte data2
	data2 = BitNeg(data1) //217
	Define byte-type variable data1, and assign it with 38, perform logical negation on data1, and 217 is returned.

12.3.12.6 BitOr

Explanation	
	It is used to perform logical disjunction (or) on byte-type data.

Return value	
	Data type: byte
	It represents the byte data obtained by performing the logical disjunction.

Definition	
	BitOr (BitData1, BitData2)
BitData1	
	Data type: byte
	The byte data 1 to be processed.
BitData2	
	Data type: byte
	The byte data 2 to be processed.

Example	
Example 1	
	VAR byte data1 = 39
	VAR byte data2 = 162
	VAR byte data3
	data3 = BitOr(data1, data2) //167
	Define the byte-type variable data1 and data2, assign them with the value of 39 and 162, respectively; perform logical conjunction on data1 and data2, and 167 is returned.

12.3.12.7 BitRSh

Explanation	
	It is used to perform the logical right shift on byte-type data.

Return value	
	Data type: byte
	It represents the byte-type data obtained by performing the right-shift operation.

Definition	
	BitLSh (BitData, ShiftSteps)
BitData	
	Data type: byte
	Byte data to be operated.
ShiftSteps	
	Data type: int
	The bits selected for the right shift, ranging from 1 to 8.

Example	
Example 1	
	VAR int right_shift = 3
	VAR byte data1 = 38
	VAR byte data2
	data2 = BiRSh(data1, right_shift) //4
	Define byte-type variable data1, and assign it with 38, perform 3 bits right shift on data1, and 4 is returned.

12.3.12.8 BitSet

Explanation	
	To set a certain bit of byte- or int-type data to 1. The bit starts from 1.

Definition	
	BitSet BitData	IntData, BitPos
BitData	
	Data type: byte
	Byte data to be operated.
IntData	
	Data type: int
	The integer data to be operated.
BitPos	
	Data type: int
	Position of the bit to be operated, ranging from 1 to 8 for byte data and 1 to 32 for int data.

Example	
Example 1	
	VAR byte data1 = 0
	BitSet data1 1 //1
	BitSet data1 2 //3
	Define byte-type variable data1 and assign it with 255, Perform BitSet on data1, set the first bit to 1, and 1 is returned, set the second bit to 1, and 3 is returned.

12.3.12.9 BitXOr

Explanation	
	It is used to perform logical exclusive or on byte-type data.

Return value	
	Data type: byte
	It represents the byte data obtained by performing the logical disjunction.

Definition	
	BitXOr (BitData1, BitData2)
BitData1	
	Data type: byte
	The byte data 1 to be processed.
BitData2	
	Data type: byte
	The byte data 2 to be processed.

Example	
Example 1	
	VAR byte data1 = 39
	VAR byte data2 = 162
	VAR byte data3
	data3 = BitOr(data1, data2) //133
	Define the byte-type variable data1 and data2, assign them with the value of 39 and 162, respectively; perform logical exclusive or on data1 and data2, and 133 is returned

12.3.13 String operations

12.3.13.1 StrFind

Explanation	
	It is used to find the position of a particular set of characters in the string from a specific location.

Return value	
	Data type: int
	It represents the location of the first matching character. If the location is not found, the length of the returned string is added by 1.

Definition	
	StrFind (Str ChPos Set [\NotInSet])
Str	
	Data type: string
	It represents the string to be searched.
ChPos	
	Data type: int
	It represents the starting position, starting from 1, if the location is off the boundary, an error is reported.
Set	
	Data type: string
	It represents the character set to be matched.
[\NotInSet]	
	Identifier, which identifies the character that cannot be matched in the character set.

Example	
Example 1	
	VAR int found
	found = StrFind("Robotics", 1, "aeiou") //2
	Matching from the first character "R", and finding the second character "o" in the character set "aeiou", return matching location 2.
	found = StrFind("Robotics", 1, "aeiou" \NotInSet) //1
	Matching from the first character "R", and finding the first character "R" is not in the character set "aeiou", return matching location 1.

12.3.13.2 StrLen

Explanation	
	It is used to obtain the length of the string.

Return value	
	Data type: int
	It represents the current string length, which is longer than or equal to 0.

Definition	
	StrLen (Str)
Str	
	Data type: string
	It represents a string that requires the calculation of string length.

Example

Example 1

VAR int num
num = StrLen("Robotics") //8
The length of the string "Robotics" is 8.

12.3.13.3 StrMap

Explanation

It is used to back up a string, all characters in it are replaced according to the specified mapping relationship. The mapped characters correspond one to one according to their position, and the unmapped characters remain the same.

Return value

Data type: string
It represents the replaced string.

Definition

StrMap (Str, FromMap, ToMap)

Str

Data type: string
It represents the original string.

FromMap

Data type: string
It represents the index of the mapping.

ToMap

Data type: string
It represents the value of the mapping.

Example

Example 1

VAR string str
str = StrMap("Robotics", "aeiou", "AEIOU") //RObOtIcs
Maps the string "Robotics", and "aeiou" is respectively mapped to "AEIOU".

Use restrictions

❯ FromMap and ToMap have to match with each other and have to be of the same length.

12.3.13.4 StrMatch

Explanation

It is used to search in a string, starting at the specified location, search for a particular format or a string, and return the matched location.

Return value

Data type: int
It represents the position of the first character of the matched string. If there is no match, the string length plus one is returned.

Definition

StrMatch (Str, ChPos, Pattern)

Str

Data type: string

It represents the string to be searched.

ChPos

Data type: int

It represents the starting position, if the location exceeds the length range of the string, an error is reported.

Pattern

Data type: string

It represents the format string to match.

Example

Example 1

VAR int found

Found = StrMatch("Robotics", 1, "bo") //3

Search from the first character for "bo" and find a match at the third position, position 3 is returned.

12.3.13.5 StrMemb

Explanation

It is used to check whether a character in a string belongs to a specified character set.

Return value

Data type: bool

True indicates that the character in the string belongs to the specified character set. Otherwise, false is returned.

Definition

StrMemb (Str, ChPos, Set)

Str

Data type: string

It represents the string to be checked.

ChPos

Data type: int

It represents the position of the character to be checked; if it exceeds the range of the string, an error is reported.

Set

Data type: string

It represents the character set to be matched.

Example

Example 1

VAR bool memb

memb = StrMemb("Robotics", 2, "aeiou") //true

The second character o is a member of the character set "aeiou" and true is returned.

12.3.13.6 StrOrder

Explanation

It is used to compare two strings and return the Boolean value.

Return value	
	Data type: bool
	When str1<=str2, returns true, otherwise, false.

Definition	
	StrOrder (Str1, Str2)
Str1	
	Data type: string
	It represents the first string value.
Str2	
	Data type: string
	It represents the second string value.

Example	
Example 1	
	VAR bool le
	le = StrOrder("FIRST", "SECOND") //true
	le = StrOrder("FIRSTB", "FIRST") //false

12.3.13.7 StrPart

Explanation	
	It is used to truncate a part of a string to generate a new string.

Return value	
	Data type: string
	It represents the truncated string, truncating a string from a specified location with a specified length.

Definition	
	StrPart (Str, ChPos, Len)
Str	
	Data type: string
	It represents the original string of a truncated string.
ChPos	
	Data type: int
	It represents the starting position, and if it exceeds the range of the string, an error is reported.
Len	
	Data type: int
	It represents the length for truncating.

Example	
Example 1	
	VAR string part
	part = StrPart("Robotics", 1, 5) //Robot
	Truncate the string for a length of 5 bits from position 1 to get "Robot".

12.3.13.8 StrSplit

Explanation	
	It is used to split a string into an array of strings by specifying a separator

Return value	
	Data type: string array
	It represents the array of strings obtained by splitting

Definition	
	StrSplit (Str [, separator])
Str	
	Data type: string
	It represents the original string to be split.
separator	
	Data type: string
	A separator. All characters in the string are considered as a separator and can be defaulted. If no separators exist, space can be considered as the default separator.

Example	
Example	
	string str_arr[4] = StrSplit("test1,test2;test3\test4", "\,;")
	The string is split into four substrings (test1 test2 test3 test4).

Use restrictions	
	➢ An error is reported when the input string is blank.
	➢ If the split results do not match the length of the defined string, an error is reported.

12.3.13.9 StrToByte

Explanation	
	StrToByte can convert a string into byte type data

Return value	
	Data type: byte
	The conversion result of a string.

Definition	
	StrToByte (Str [, trans])
Str	
	Data type: string
	The string to be converted.
trans	
	Data type: enumeration
	Indicates the mathematical binary format of the string. Available parameters include \Bin (binary), \Okt (octal), \Hex (hexadecimal), \Char (character), and the default (no parameter, decimal)

Example	
Example 1	
	Byte NumBin = StrToByte("10", \Bin)
	Byte NumOkt = StrToByte("10", \Okt)
	Byte NumBin = StrToByte("10")
	Byte NumHex = StrToByte("10", \Hex)
	The string "10" is converted to byte numbers in binary, octal, decimal, and hexadecimal in order, and the results are:
	2, 8, 10, 16.
Example 2	
	Byte NumChar = StrToByte("0", \Char)

The character "0" is converted to 48 according to the conversion relationship between characters and ASCII.

Use restrictions

➤ An error will be reported when the input string does not conform to the specified data format.

12.3.13.10 StrToDouble

Explanation

StrToDouble can convert a string into double type data

Return value

Data type: double
The conversion result of a string.

Definition

StrToDouble (Str)

Str

Data type: string
The string to be converted.

Example

Example

Double NumDouble = StrToDouble("3.1415926")
Convert string "3.1415926" into double type data.

Use restrictions

➤ An error will be reported when the input string does not conform to the specified data format.

12.3.13.11 StrToInt

Explanation

StrToInt can convert a string into Int type data

Return value

Data type: Int
The conversion result of a string.

Definition

StrToDouble (Str)

Str

Data type: string
The string to be converted.

Example

Example

Int NumInt = StrToInt("99")
Convert string "99" into Int type data.

Use restrictions

➤ An error will be reported when the input string does not conform to the specified data format.

12.3.14 Operators

8.3.11.1 Basic operators

Arithmetic operators

Arithmetic operators include:

Operator	Application
+	Plus
-	Minus
*	Multiply
/	Divide
%	Modular arithmetic
--	Decrement
++	Increment

The arithmetic operators support the operation of data defined as int or double type. The examples for arithmetic operators are as follows:

Example 1

VAR int a = 1
VAR int b = 2
VAR int c = -b //Negate
VAR int ac = a * c //Multiplication

Example 2

The two operators ++ and --, also known as unary operators, are operators that operate on an operand. RL does not distinguish between pre and post increment or decrement:
x = n++   //Means to add n by 1 and assign the n value to x
x = --n   //Means to subtract n by 1 and assign the new value to x

Logical operators

Logical operators support the operation of the basic data types, including

Operator	Application
&&	Logical conjunction
||	Logical disjunction
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
!	Take logical negation

Logic and && expressions are true if the results on both sides are true, and the logic or || expression is true if one of the conditions of the two sides is true.

Example 1

The examples for other logical operators are as follows:
VAR int res = 1
while(res < 3) //Compare to determine whether res is less than 3
    res++
endwhile
di5 = !di6 // Take logical negation
VAR int counter = 4
while(di7&&di8) /Calculate logical conjunction

```
if(counter == 5) //Whether it equals to
        break
    endif
endwhile
```

Assignment operators

Assignment operators include:

Operator	Application
=	Assignment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment

The examples for assignment operators are as follows

VAR int num1 = 3

VAR int num2 = 4

num1 += num2 //Equivalent to num1 = num1 + num2, then num1 = 7.

num1 -= num2 //Equivalent to num1 = num1 – num2, then num1 = -1.

num1 *= num2 //Equivalent to num1 = num1 * num2, then num1 = 12.

num1 /= num2 //Equivalent to num1 = num1 / num2, then num1 = 0.

num1 %= num2 //Equivalent to num1 = num1 % num2, then num1 = 3.

Other operators

Operator	Application
()	Parentheses
.	Dot operator

The examples for the operators are as follows:

Example 1

VAR int num = arr[1]   //Assign the first element of the array to num

VAR int num2 = (1+2)*3   //Using parentheses can change the order of operations, the value of num2 here is 9

Example 2

Define a robtarget variable pt1

pt1.trans.x = 200 // Change the x coordinate of the pt1 point to 200 using the "." operator

Use restrictions

➢ The "." operator does not support modifications to the A, B, C members of robtarget variables.

8.3.11.2 Operation priority

Priority	Operator	Use form	Combination direction
1	( )	(Expression)/function name (formal parameter list)	
	.	Variable name.	
2	-	-Expression	From right to left
	++	++ Variable name/Variable name ++	
	--	--Variable name/Variable name --	
	!	!Expression	
3	/	Expression / Expression	From left to right
	*	Expression * Expression	
	%	Integer expression / Integer expression	
4	+	Expression + Expression	From left to right
	-	Expression - Expression	
5	>	Expression > Expression	From left to right
	>=	Expression >= Expression	
	<	Expression < Expression	
	<=	Expression <= Expression	
6	==	Expression == Expression	From left to right
	!=	Expression != Expression	
7	&&	Expression && Expression	From left to right
8	||	Expression || Expression	From left to right
9	=	Variable = Expression	From right to left
	/=	Variable /= Expression	
	*=	Variable *= Expression	
	%=	Variable % = Expression	
	+=	Variable += Expression	
	-=	Variable -= Expression	

12.3.15 Clock commands

12.3.15.1 ClkRead

Explanation

It is used to read the value of the clock.

Return value

Data type: double

Returns the time interval between the stop time of the clock or the current time and the start of the clock. The accuracy is 0.001s.

Definition	
	ClkRead (Clock)
Clock	
	Data type: clock
	Name of the clock.

Example	
Example 1	
	VAR clock clock1
	ClkStart clock1
	ClkStop clock1
	VAR double interval=ClkRead(clock1)
	interval stores the time interval between start and stop of clock1.

12.3.15.2 ClkReset

Explanation	
	It is used to reset a clock.
	ClkReset guarantees that the count is 0 before using a clock.

Definition	
	ClkReset Clock
Clock	
	Data type: clock
	Name of the clock.

Example	
Example 1	
	VAR clock clock1
	ClkReset clock1
	Reset clock1.

12.3.15.3 ClkStart

Explanation	
	It is used to start a clock.
	When a clock starts, it will continue to count until the clock stops or the program resets. The clock will continue to operate after the program stops or the robot is powered off.

Definition	
	ClkStart Clock
Clock	
	Data type: clock
	Name of the clock.

Example	
Example 1	
	VAR clock clock1
	ClkStart clock1
	Declare clock1, and start clock1.

12.3.15.4 ClkStop

Explanation	
	It is used to stop a clock.
	When the clock stops, it stops counting. After the clock stops, it can be read for the interval, restarted, or reset.

Definition	
	ClkStop Clock
Clock	
	Data type: clock
	Name of the clock.

Example	
Example 1	
	VAR clock clock1
	ClkStart clock1
	…
	ClkStop clock1
	Stop clock1.

12.3.16 Advanced commands

12.3.16.1 RelTool

Explanation	
	It is used to translate or rotate the spatial position in the tool frame as specified by the current command.
	There are two main differences from Offs:
	➢ Offs is the offset relative to the work object frame, and RelTool is the offset relative to the tool frame;
	➢ The Offs function does not support offsets of orientations, but RelTool does.

Return value	
	Data type: robtarget
	Return the new pose after the offset.

Definition	
	RelTool(Point, XOffset, YOffset, ZOffset, Rx, Ry, Rz [, Tool, Wobj])
Point	
	Data type: robtarget
	The point to be offset, or the initial point of the offset command.
XOffset	
	Data type: double
	Offset in the x-direction of the tool frame.
YOffset	
	Data type: double
	Offset in the y-direction of the tool frame.
ZOffset	
	Data type: double
	Offset in the z-direction of the tool frame.

Rx

 Data type: double

 The rotation angle around the x-axis of the tool frame.

Ry

 Data type: double

 The rotation angle around the y-axis of the tool frame.

Rz

 Data type: double

 The rotation angle around the z-axis of the tool frame.

Tool

 Data type: tool

 Contain tool frame information describing the Point position

Wobj

 Data type: wobj

 Contain work object frame information describing the Point position

Example	
Example 1	

p2=RelTool(p1,100,0,30,20,0,0)
Since no tool and work object is specified, tool0 and wobj0 are used by default. Offset point p1 by 100 mm in the x-direction, 0 mm in the y-direction, and 30 mm in the z-direction on the work object frame, and then rotate 20 degrees around the x-axis. Last assign the new target point position to p2.

Example 2

p2=RelTool(p1,100,0,30,20,0,0, tool5, wobj6)
Offset point p1 by 100 mm in the x-direction, 0 mm in the y-direction, and 30 mm in the z-direction on the wobj6 work object frame, and then rotate 20 degrees around the x-axis. Last assign the new target point position to p2.

Example 3

MoveL RelTool(p1, 100,0,30,20,0,0), v4000, fine, tool2, wobj4
RelTool is used along with the Move command. As no tool or work object frame is specified, the tool and wobj of the Move command will be used. Offset point p1 by 100 mm in the x-direction, 0 mm in the y-direction, and 30 mm in the z-direction on the wobj4 work object frame, and then rotate 20 degrees around the x-axis. Last assign the new target point position to p2.

> **ℹ** Notes
>
> Auxiliary programming is not supported for the optional parameters (Tool and Wobj) of this command.

12.3.16.2 Offs

Explanation

 The position offset function, which is used to offset a point in the work object frame specified in the current command by a distance and return the position value of a new point.
 The translation offset is represented by x, y, and z, and the orientation rotation offset is represented by Rx, Ry, and Rz.

Return value

 Data type: robtarget
 The new pose after the offset.

Definition

Offs (Point, XOffset, YOffset, ZOffset [, Rx, Ry, Rz])

Point

Data type: robtarget
The point to be offset, or the initial point of the offset command.

XOffset

Data type: double
Offset in the x-direction of the work object frame.

YOffset

Data type: double
Offset in the y-direction of the work object frame.

ZOffset

Data type: double
Offset in the z-direction of the work object frame.

Rx

Data type: double
The rotation angle around the x-axis of the tool frame.

Ry

Data type: double
The rotation angle around the y-axis of the tool frame.

Rz

Data type: double
The rotation angle around the z-axis of the tool frame.

Example

Example 1

p11=Offs(p10,100,200,300)

Have the point p10 offset 100 mm in the x-direction, offset 200 mm in the y-direction, offset 300 mm in the z-direction of the work object frame, and assign the position of the new target point to p11.

> **i** Notes
>
> This command does not support auxiliary programming for the moment.

12.3.16.3 ConfL On/Off

Explanation

There is a set of conf parameters (cf1-7, cfx) in the xMate Cartesian frame. The conf data corresponding to the Cartesian coordinate points manually changed or written by the user may be incorrect, which makes it impossible for the controller to resolve the path of the target point. But in some scenarios, the user cares only about the robot's TCP location rather than the orientation. In this case, ConfL Off can be used to remove conf limitations and the controller can try to compute a feasible set of conf parameters (may not be calculated, resulting in failure of motion command)

Example

Example 1

p1.trans.x = ….
MoveJ p1, v1000 ….
Only the frame is modified, not the cf parameters. This command is likely to cause the execution to fail

…

Example 2	ConfL Off MoveJ p1, v1000 …. Disable conf check. The robot can move to point p1, but the orientation is uncertain ConfL On Enable conf check
Notes	• The conf restriction is turned on (ConfL On) by default for the "Move to" function on the HMI point position list interface; • The conf restriction is turned off (ConfL Off) by default for RL programming.

12.3.16.4 VelSet

Explanation	The VelSet command allows for adjusting maximum motion speed for smoother motion when the robot is handling fragile objects. Instead of being constant, the maximum velocity of each joint keeps changing with load, body orientation, and other factors when the robot is moving. The VelSet command scales the maximum velocity capability curve for a specific task path, and the scaled maximum velocity capability curve is also a changing curve.
Definition gain	VelSet gain Data type: int The maximum velocity capacity is specified in percentage, ranging from 1% to 100%, where 100% means the maximum acceleration. The robot reports an error when going over the limit.
Example Example 1	VelSet 50 Set the maximum velocity capability to half of the robot's default maximum velocity.
Notes	1. The VelSet command only affects the motion commands of the corresponding RL project, instead of JOG, move-to, rapid motion, and other non-project functions. 2. The VelSet function will interrupt the turning zone. Please do not insert VelSet commands between the motion commands that require a turning zone. 3. The difference between the VelSet command and the program running rate adjustment slide: the program running rate adjustment slide modifies the user's expected velocity, for example, motion command V4000, under 50% slide control, equals a user's expected velocity of V2000. But if the robot is at its limits, the actual maximum velocity of this motion command is only V1000, then the actual motion velocity of the robot does not change regardless of whether the velocity slide is at 50% or 100%, because both V2000 and V4000 are above V1000. Changing the expected velocity during this range will not impact the actual execution velocity; on the contrary, VelSet 50 does not change the user's expected velocity but reduces the actual maximum velocity of the motion command by 50% during the motion planning process. Under the same motion command, the actual robot motion velocity will be cut to half from V1000 to V500. The user should identify the difference between these two functions. 4. Acceleration automatically reverts to the default (100%) during the following operations: ➢ RL program is reset manually (PP to Main) ➢ A new RL program is loaded

12.3.16.5 AccSet

Explanation	
	The AccSet command allows for adjusting acceleration and deceleration for smoother movement when the robot is handling fragile objects.
Definition	
	AccSet acc, ramp
acc	
	Data type: int The acceleration and deceleration are specified as a percentage of the system preset value, ranging from 30% to 100%, where 100% means the maximum acceleration, beyond which the robot will stop and report an error.
ramp	
	Data type: int The Jerk is specified as a percentage of the system preset value, ranging from 10% to 100%, where 100% means the maximum jerk, beyond which the robot will stop and report an error.
Example	
Example 1	
	AccSet 50,15 Acceleration and jerk are set to half of the default.
Notes	
	Acceleration automatically reverts to the default (100%) during the following operations: ➢ RL program is reset manually (PP to Main) ➢ A new RL program is loaded

12.3.16.6 EulerToQuaternion

Explanation	
	It is used to convert Euler angle to quaternion.
Return value	
	It represents the conversion result, 0 means successful, others mean abnormal.
Parameter	
	EulerToQuaternion (type,A,B,C,q1,q2,q3,q4)
type	
	Euler angle order type, including EULER_XYZ and EULER_ZYX.
A,B,C	
	It represents the Euler angle to be converted. Data type: double
q1~q4	
	It represents the quaternion obtained from the conversion. Data type: double

12.3.16.7 QuaternionToEuler

Explanation	
	It is used to convert a quaternion to an Euler angle.

Return value	
	It represents the conversion result, 0 means successful, others mean abnormal.

Parameter	
	QuaternionToEuler (type,q1,q2,q3,q4,A,B,C)
type	
	Euler angle order type, including EULER_XYZ and EULER_ZYX.
q1~q4	
	It represents the quaternion to be converted.
	Data type: double
A,B,C	
	It represents the Euler angle obtained from the conversion.
	Data type: double

12.3.16.8 GetEndtoolTorque

Explanation	
	Get the end-effector tool torque information in the tool frame specified by the current command, which is used for the force control task.

Return value	
	Data type: TorqueInfo
	End-effector torque information

Definition	
	GetEndtoolTorque(tool, wobj [, type])
tool	
	Data type: Tool parameter
	This parameter should provide the current tool used by the robot, since the handheld load may change at any time when the robot is working
wobj	
	Data type: Work object parameter
	This parameter should provide the current work object used by the robot, since the handheld load may change at any time when the robot is working
type	
	Data type: int enumeration
	0 Torque of the end-effector relative to the world frame
	1 Torque of the end-effector relative to the flange frame
	2 Torque of the end-effector relative to the TCP

Example	
Example 1	
	TorqueInfo tmp_info = GetEndtoolTorque(tool1, wobj1)
	Obtain the information architecture of the torque applied to the tool at the end-effector of the robot in the case of tool1 wobj1
	print(tmp_info.joint_torque.measure_torque)
	print(tmp_info.joint_torque.external_torque)

Print the measured force and external force of each axis

print(tmp_info.cart_torque.m_torque)
Print Cartesian space torque

print(tmp_info.cart_torque.m_force[0])
print(tmp_info.cart_torque.m_torque[0])
Print information of force and torque in X direction

12.3.16.9 MotionSup

Explanation	
	Used to turn on and off Collision Detection
Definition	
	MotionSup type [, level]
type	
	Data type: keyword
	on to turn on, off to turn off
level	
	Data type: string
	Additional parameter for MotionSup On, used to modify the collision detection sensitivity
	"High" for high collision sensitivity
	"Medium" for medium collision sensitivity
	"Low" for low collision sensitivity
Example	
Example 1	
	MotionSup On
	//... Other commands
	MotionSup Off
	Turn on Collision Detection and then execute other commands. When the commands are executed, use MotionSup Off to turn off Collision Detection
Example 2	
	MotionSup On, "High"
	Turn on Collision Detection and set the detection sensitivity to high

12.3.16.10 MotionSupPlus

Explanation	
	MotionSupPlus (Motion Supervision Plus) is used to adjust the robot's joint collision detection sensitivity in the RL program at any time.
	MotionSupPlus x1,x2,x3,x4,x5,x6,x7, where x1 to x7 represent the collision detection sensitivity thresholds in Nm for joints 1-7, respectively.
Example	
Example 1	
	MotionSupPlus 5,20,7,20,6,20,5
	Indicates the sensitivity thresholds of the 7 joints to be 5, 20, 7, 20, 6, 20, 5(Nm), respectively.
	Note: For 6-axis robots, 7 parameters should be set too, where the first 6 parameters correspond to joints 1-6.
	This command is available for cobots and six-axis industrial robots, but not three- and four-axis industrial robots.

12.3.16.11 CONNECT (expired)

Explanation	
	To associate the interrupt identifier with the TRAP scope.
	Interrupts are defined by customizing an interrupt event and assigning an interrupt identifier.
	Therefore, when the event occurs, the TRAP scope executes.

Definition	
	CONNECT Interrupt WITH TRAP
Interrupt	
	Data type: intnum
	An interrupt descriptor.
	The interrupt descriptor must be a global variable.
TRAP	
	Data type: string
	TRAP scope name.

Example	
Example 1	
	VAR intnum test_int
	PROC main()
	CONNECT test_int WITH test_TRAP
	ISignalDI di1, 1 , test_int
	The interrupt description test_int is connected to the TRAP scope test_TRAP. When di1 goes high, an interrupt will be generated. In other words, when the signal di1 goes high, the scope test_TRAP will be executed.

12.3.16.12 BreakLookAhead

Explanation	
	This command informs the control system to cancel the lookahead and force the cancellation of the turning zone between the previous motion command and the next motion command. The robot TCP will move to the target point position of the previous motion command and then move to the next point without the turning zone. The program pointer will also wait for the TCP to move to the target point position of the previous motion command before continuing the lookahead scan.

Definition	
	BreakLookAhead
	The command includes no parameters and no return value.

Example	
Example 1	
	MoveL P1,v1000,z50,tool0
	BreakLookAhead
	MoveL P2,v1000,z50,tool0
	MoveL P3,v1000,z50,tool0
	1) The turning zone of point P1 is set to z50. Because of the BreakLookAhead command, the lookahead and the turning zone will be canceled, and the robot TCP will move exactly to point P1 and then to P2. There is no BreakLookAhead command between P2 and P3, so the robot will look ahead at P2 and pass the z50 turning zone before moving to P3.
	2) The BreakLookAhead command has the same effect as the wait 0 command.

12.3.16.13 GetRobotMaxLoad

Explanation	
	Get the maximum load value of the current robot model.

Definition	
	Ret = GetRobotMaxLoad()
Ret	
	Data type: int
	Maximum payload

Example	
Example 1	
	int maxload = GetRobotMaxLoad()
	print(maxload)
	With xMate 7 as an example, return 7.

12.3.16.14 GetRobotState

Explanation	
	Get the current operating state of the control system. Use the 4-byte bit information to represent the state of the control system, including fault, emergency stop, safety gate, operation mode, servo mode, and motion state, as shown in following table.

No.	State bits	Meaning
1	Byte[1].bit[1]	1: Control system is not authorized
2	Byte[1].bit[2]	1: Control system recoverable faults
3	Byte[1].bit[3]	1: Control system fatal error
4	Byte[1].bit[4]	1: Servo system failure
5	Byte[1].bit[5]	1: Servo system fatal failure
6	Byte[1].bit[6]	1: Emergency stop
7	Byte[1].bit[7]	1: Safety gate stop
8	Byte[1].bit[8]	Reserved
9	Byte[2].bit[1]	Power-on state, 0: motor is not powered on; 1: motor is powered on
10	Byte[2].bit[2]	Robot motion state, 0: idle; 1: in motion
11	Byte[2].bit[3]	Operation mode, 0: manual mode; 1: automatic mode
12	Byte[2].bit[4]	Servo mode, 0: position mode; 1: torque mode
13	Byte[2].bit[5]	Reserved
14	Byte[2].bit[6]	Reserved
15	Byte[2].bit[7]	Reserved
16	Byte[2].bit[8]	Reserved
17	Byte[3]	Reserved
18	Byte[4]	Reserved

Definition	
	Ret = GetRobotState()
Ret	
	Data type: byte array
	Use four-byte types to represent the robot state.

Example	
Example 1	
	byte st[4] = GetRobotMaxLoad()
	print(st)
	Return {0,5,0,0}. According to the table, the current state is: no fault, motor powered on, automatic mode, servo is in position mode.

12.3.16.15 AutoIgnoreZone true/false

Explanation

Used to specify whether to allow the control system to automatically ignore the turning zone.
Definition

AutoIgnoreZone true/false

AutoIgnoreZone true: allow the control system to automatically ignore the turning zone;

AutoIgnoreZone false: do not allow the control system to automatically ignore the turning zone;
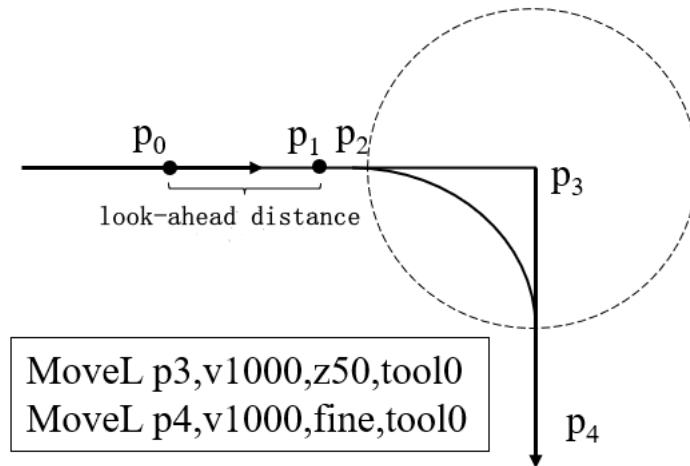
As shown above: The robot runs two MoveL commands with a z50 turning zone in between. During the motion, the robot needs lookahead from its current position for smooth and safe motion. For example, when the robot moves to p_0, it looks ahead to P_1. In this process, the control system pre-processes the information between the two points.

As the robot moves forward, the lookahead end point also moves forward. At a certain point, the lookahead end point p_1 coincides with p_2, the start point of the turning zone. If the control system has received the second motion command, it can generate a turning zone properly and control the robot to move along the predetermined trajectory; if the control system fails to receive the second motion command, it cannot generate the turning zone, and it will process the turning zone according to the AutoIgnoreZone command status. See below for the logic:

AutoIgnoreZone true: Instead of waiting for the second motion command, the control system will cancel the turning zone and control the robot to move directly toward P_3.

AutoIgnoreZone false: The control system will wait for the second motion command, during which the robot will slow down until the turning zone trajectory is generated. If the robot fails to receive the second motion command when reaching P_2, the robot will stop moving and report an error through HMI.

The failure of the robot to receive the second motion command timely is often a result of too many non-motion commands between two motion commands, e.g.:

```
MoveL p3,v1000,z50,tool0
For(int i from 1 to 10000)
    printf("i = %d\n", i)
endfor
MoveL p4,v1000,fine,tool0
```

Many print commands are added between two motion commands, and it takes a long time for the control system to receive the second motion command after the first one is processed.
Default: AutoIgnoreZone true

Example

Example 1

AutoIgnoreZone true
MoveL p3,v1000,z50,tool0
MoveL p4,v1000,fine,tool0
Allow the control system to automatically ignore the turning zone

Example 2

AutoIgnoreZone false
MoveL p3,v1000,z50,tool0
MoveL p4,v1000,fine,tool0
 Do not allow the control system to automatically ignore the turning zone

12.3.16.16 MotionWaitAtFinePoint true/false

Explanation

When the robot is stationary and the user clicks Start, the control system will look ahead a certain distance according to the lookahead parameter (see Chapter 10.2.2 Lookahead mechanism) before starting the robot. This command sets whether the robot starts moving immediately when the lookahead coincides with a fine point.
A fine point refers to the target point without a turning zone, i.e. a target point with the turning zone parameter set to fine.
MotionWaitAtFinePoint true: The control system controls the start of the robot strictly according to the lookahead parameters (see Chapter 10.2.2 Lookahead mechanism). The robot only starts to move when the lookahead distance reaches the set value of the lookahead parameter or the lookahead of all motion commands is completed. In this state, the control system can guarantee the set lookahead distance.
MotionWaitAtFinePoint false: The control system does not strictly follow the lookahead parameters, and the robot starts moving immediately when the lookahead coincides with the fine point. In this state, the robot can still start smoothly when the program logic gets extremely complicated, but the lookahead distance cannot be guaranteed.
Default: MotionWaitAtFinePoint false

Example

Example 1

MotionWaitAtFinePoint true
MoveL p1,v1000,fine,tool0
MoveL p2,v1000,fine,tool0
MoveL p3,v1000,fine,tool0
MoveL p4,v1000,fine,tool0
MoveL p5,v1000,fine,tool0
When the control system looks ahead to p_1, it does not start the robot immediately, but checks whether the current lookahead distance has reached the set length before deciding whether to start the robot.

Example 2

MotionWaitAtFinePoint false
MoveL p1,v1000,fine,tool0
MoveL p2,v1000,fine,tool0
MoveL p3,v1000,fine,tool0
MoveL p4,v1000,fine,tool0
MoveL p5,v1000,fine,tool0
When the control system looks ahead to p_1, it immediately starts the robot, instead of checking whether the current lookahead distance has reached the set length.

12.3.17 Function commands

12.3.17.1 CRobT

Explanation	
	It is used to get the robot pose. When using this function, you need to give the names of the tool and the work object. Return the pose of the specified tool frame, the current axis configuration information, and the external axis position. When using CRobT, the robot should be in the stop state, i.e. the turning zone of the motion command before CRobT should be set as fine.
Return value	
	Data type: robtarget Return the current robot position, orientation, axis configuration data, and external axis information.
Definition	
	CRobT(Tool, Wobj)
Tool	
	Data type: tool The tool used when calculating the position.
Wobj	
	Data type: wobj The work object used when calculating the position.
Example	
Example 1	
	p2 = CRobT(tool1, wobj2)

12.3.17.2 CJointT

Explanation	
	CJointT is used to read the current angle of the robot axes and external axes. When using CJointT, the robot should be in the stop state, i.e. the turning zone of the motion command before CRobT should be set as fine.
Return value	
	Data type: jointtarget Rotation axis unit: degree; Linear axis unit: mm Return the current angle value of the robot axes and the external axes
Definition	
	CJointT () Data type: function
Example	
Example 1	
	VAR jointtarget j2 j2 = CJointT ()

12.3.17.3 CalcJointT

Explanation	
	It is used to calculate the corresponding joint angle based on the specified robtarget variable.

Return value	
	Data type: jointtarget
	Return the positions of joint angle and external axes corresponding to the input position.
	The joint angle is in Degrees, the external axis of the straight line is in millimeters (mm), and the rotation of the external axis is in Degrees.

Definition	
	CalcJointT (Rob_Target, Tool, Wobj)
Rob_Target	
	Data type: robtarget
	The specified Cartesian space target point. Please note that the tool and work object used in the definition of this point should be consistent with the tool/work object used in the CalcJointT command, otherwise, it may lead to results error.
Tool	
	Data type: tool
	The tool to be used when calculating the joint angle. Note that it needs to be the same as the one used when defining the robtarget used.
Wobj	
	Data type: wobj
	The work object to be used when calculating the joint angle. Note that it needs to be the same as the one used when defining the robtarget used.

Example	
Example 1	
	jpos2 = CalcJointT(pt1, tool1,wobj2)
	Calculate the joint angle corresponding to tool1 when it reaches pt1, and assign it to jpos2. pt1 is defined under the work object wobj2.

12.3.17.4 CalcRobt

Explanation	
	It is used to calculate the corresponding Cartesian space pose based on the specified joint angle.

Return value	
	Data type: robtarget
	Return the Cartesian space pose of a given joint angle.

Definition	
	CalcRobt (Joint_Target, Tool, Wobj)
Joint_Target	
	Data type: jointtarget
	The given joint angle for calculating Cartesian space pose.
Tool	
	Data type: tool
	The tool used when calculating Cartesian space pose.

Wobj	
	Data type: wobj
	The work object used when calculating the Cartesian space pose.

Example	
Example 1	
	pt1 = CalcRobT (jpos1, tool2,wobj1)
	Calculate the Cartesian space pose according to the joint angle jpos1 and assign it to pt1.
	pt1 is the pose described by the tool frame tool2 in the work object frame wobj1.

12.3.17.5 Print

Explanation	
	It is used to print and output the user-defined content to the teach pendant, and the user can then use this function to debug the program.
	The input parameters of the Print function are special, the number of input parameters is unlimited, but there must be at least one, and each parameter must be a defined variable or a constant.
	The system converts these variables into strings and concatenates them in series, and finally outputs them to the debug window of the program editor.

Definition	
	Print (var1, var2, ...)

Example	
Example 1	
	counter = 0
	while(true)
	counter++
	Print("counter = ",counter)
	endwhile
	After the program is executed, the HMI's program debug window will print the following information:
	counter = 1
	counter = 2
	counter = 3
	counter = 4
	……

> **ℹ Notes**
>
> When you need to output a string, you can use double quotation marks "" to include the characters you want to display, but nested double quotation marks in double quotation marks are not supported.

12.3.17.6 PoseMult

Explanation	
	PoseMult is used to calculate the product of two pose changes.

Definition

pose3 = PoseMult(pose1, pose2)

Parameter explanation: pose1 and pose2 are input of the pose type, and pose3 is the return value of the pose type.

Example

pose1 represents the pose of frame 1 relative to frame 0, and pose2 the pose of frame 2 relative to frame 1. Pose3, the pose of frame 2 relative to frame 0 can be calculated through the following method:

VAR pose pose1

VAR pose pose2

VAR pose pose3

...

pose3 = PoseMult(pose1, pose2)

12.3.17.7 PoseInv

Explanation

PoseInv is used to calculate the inversion of a pose change.

Definition
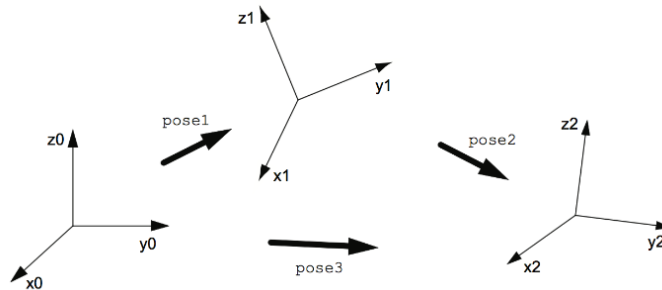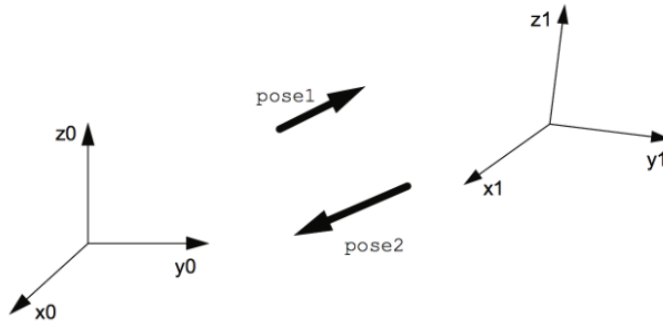
pose2 = PoseInv(pose1)

Parameter explanation: pose1 is input of the pose type, and pose2 is the return value of the pose type.

Example

pose1 represents the pose of frame 1 relative to frame 0, and pose 2 the pose of frame 0 relative to frame 1.

If pose1 is known, pose2 can be calculated through the following method:

VAR pose pose1

VAR pose pose2

...

pose2 = PoseInv(pose1)

12.3.17.1 GetRobAbc

Explanation	
	Get the Euler angle orientation ABC of the Cartesian space point P; the rotation sequence: the initial frame (the work object frame selected in the motion command) first rotates around its own X axis, then around its Y axis, and last around its Z axis

Definition	
	double db_arr[3] = GetRobABC(Point [, A, B, C])
Point	
	Data type: Cartesian point position
	The Cartesian point position used when calculating the position.
A, B, C	
	Data type: double
	The return value of the Euler angle orientation for the Cartesian point position.

Return value	
	Data type: double-type three-dimensional array
	The return value of the Euler angle orientation for the Cartesian point position.

Example 1	
	point0 is a Cartesian point position variable. To convert the Euler angle orientation of the variable to a Double variable of RL, use the following RL command
	VAR double Rob_A
	VAR double Rob_B
	VAR double Rob_C
	// Assign the Euler angle of point0 to Rob_A|B|C
	GetRobAbc(point0, Rob_A, Rob_B, Rob_C)

Example 2	
	point0 is a Cartesian point position variable. To generate an array of temporary variables to store the Euler angles of the Cartesian point position, use the following RL command
	double db_arr[3] = GetRobAbc(point0)

12.3.17.2 SetRobAbc

Explanation	
	Get the orientation of the Cartesian space point P based on the Euler angles ABC entered; the rotation

sequence: the initial frame (the work object frame selected in the motion command) first rotates around its own X axis, then around its Y axis, and last around its Z axis.

Definition	
	SetRobABC(Point , A, B, C)
Point	
	Data type: Cartesian point position
	The Cartesian point position whose orientation to be modified.
A, B, C	
	Data type: double
	Set the Euler angle orientation for the Cartesian point position, in °.

Example 1	
	point0 is a Cartesian point position variable. Set the Euler angles of the point to 30°, 60°, and 90°.
	SetRobAbc(point0, 30, 60, 90)

12.3.17.3 RotRobAbc

Explanation	
	Rotate the Euler angles from the existing orientation of the Cartesian space point P based on the Euler angles ABC entered; the rotation sequence: the initial frame (orientation of the point P) first rotates around its own X axis, then around its Y axis, and last around its Z axis. The input angles ABC are added to the existing Euler angles.

Definition	
	RotRobABC(Point , A, B, C)
Point	
	Data type: Cartesian point position
	The Cartesian point position whose orientation to be modified.
A, B, C	
	Data type: double
	Set the Euler angle orientation for rotating the Cartesian point position, in °.

Example 1	
	point0 is a Cartesian point position variable. Rotate the point position around X, Y, and Z to 30°, 60°, 90°.
	RotRobAbc(point0, 30, 60, 90)

12.3.18 Register commands

12.3.18.1 ReadRegByName

Explanation	
	Reads the value of the corresponding register according to the register name

Definition	
	ReadRegByName(RegData, Value)
RegData	
	Data type: Readable register variable
	Setup -> Communication -> Register interface function, register variable.

Value	
	Data type: bool/int/double
	The register data will be written into Value, and if the register variable type mismatches with the interpreter variable, the format will be converted automatically

Example	
Example 1	
	int tmp_int
	ReadRegByName(modbus_int_read[6], tmp_int)
	Read the data named modbus_int_read with subscript 6 into tmp_int variable

12.3.18.2 WriteRegByName

Explanation	
	Reads the value of the corresponding register according to the register name

Definition	
	WriteRegByName(RegData, Value)
RegData	
	Data type: writable register variable
	Setup -> Communication -> Register interface function, register variable.
Value	
	Data type: bool/int/double
	The Value will be written into the register, and if the register variable type mismatches with the interpreter variable, the format will be converted automatically

Example	
Example 1	
	WriteRegByName(modbus_int_write[6], 200)
	Write the data of INT 200 to the register corresponding to modbus_int_write[6].

12.3.19 End-effector commands

12.3.19.1 JodellGripInit

Explanation	
	Initialization command of Jodell electric gripper

Definition	
	JodellGripInit ID,wait_time
ID	
	Data type: Int variable
	Establish communication, initialize Jodell electric gripper, parameter ID.
Wait_time	
	Data type: Int variable
	Wait for the initialization to complete, wait time threshold, report error on timeout, in s.

12.3.19.2 JodellGripMove

Explanation	
	Motion command of Jodell electric gripper

Definition	
	JodellGripMove ID,Pos,Vel,Trq

ID

    Data type: Int variable
    The gripper ID that controls the movement of the gripper.

Pos

    Data type: Int variable
    Target position, unitless, range 0-255.

Vel

    Data type: Int variable
    Electric gripper velocity, unitless, range 0-255.

Trq

    Data type: Int variable
    Force detected by electric gripper operation, unitless, range 0-255.

12.3.19.3 JodellGripStatus

Explanation	
	Obtain the status of Jodell electric gripper

Definition	
	JodellGripStatus ID,Pos,Vel,Trq,Contact
ID	
	Data type: Int variable The gripper ID that obtains the movement status of the gripper.
Pos	
	Data type: Int variable Obtain the electric gripper's current position, unitless, range 0-255.
Vel	
	Data type: Int variable Obtain the electric gripper's velocity, unitless, range 0-255.
Trq	
	Data type: Int variable Obtain the electric gripper's torque, unitless, range 0-255.
Contact	
	Data type: Int variable Obtain the electric gripper's state, unitless, range 0-255, where bit6-7 indicate whether the electric gripper detects an object.

Bit	Name	Value/Description
0	gAct	0: the electric gripper is being reset; 1: the electric gripper is in the enabling state
2	gMode	0: the parameter control mode; 1: the parameterless control mode
3	gGTO	0: stop; 1: moving to the target position
4-5	gSTA	0: the electric gripper is being reset or in the inspection state; 1: being activated; 2: not used; 3: activation completed
6-7	gOBJ	0: fingers are moving to the specified position; 1: fingers stop due to contact with an object when opening to reach the specified position; 2: fingers stop due to contact with an object when closing to reach the specified position; 3: fingers reach the specified position, but no object is detected.

12.3.19.4 JodellSuckInit

Explanation	
	Initialization command of Jodell suction cup

Definition	
	JodellSuckInit ID
ID	
	Data type: Int variable Initialize the suction cup of this ID and detect if the suction cup of this ID is connected correctly.

12.3.19.5 JodellSuckSet

Explanation	
	The command for Jodell suction cup to operate. When this command is given, the suction cups immediately start operating according to the set parameters.

Definition	
	JodellSuckSet ID,CH1_enable,CH1_VacMin, CH1_VacMax,CH1_Waittime,CH2_enable,CH2_VacMin,CH2_VacMax,CH2_Waittime
ID	Data type: Int variable The ID of the suction cup being controlled.
CH1_enable	Data type: Int variable Whether the first channel of the suction cup is working or not. 1: working; 0: not working.
CH1_VacMin	Data type: Int variable The minimum vacuum level of the first channel of the suction cup, range 0-255. 0 means pure vacuum, and a value over 100 means releasing the suction cup; stop pumping when the actual vacuum level is lower than this threshold;
CH1_VacMax	Data type: Int variable The maximum vacuum level of the first channel of the suction cup, range 0-255. 0 means pure vacuum, and a value over 100 means releasing the suction cup; start pumping when the actual vacuum level is higher than this threshold;
CH1_Waittime	Data type: Double variable Timeout value of the first channel of the suction cup;
CH2_enable	Data type: Int variable Whether the second channel of the suction cup is working or not. 1: working; 0: not working.
CH2_VacMin	Data type: Int variable The minimum vacuum level of the second channel of the suction cup, range 0-255. 0 means pure vacuum, and a value over 100 means releasing the suction cup; stop pumping when the actual vacuum level is lower than this threshold;
CH2_VacMax	Data type: Int variable The maximum vacuum level of the second channel of the suction cup, range 0-255. 0 means pure vacuum, and a value over 100 means releasing the suction cup; start pumping when the actual vacuum level is higher than this threshold;
CH2_Waittime	Data type: Double variable Timeout value of the second channel of the suction cup;

12.3.19.6 JodellSuckStatus

Explanation	
	Obtain the status of Jodell suction cup

Definition	
	JodellSuckStatus ID,Vac1,Contact1,Time_Err1,Vac2,Contact2,Time_Err2
ID	Data type: Int variable The ID of the suction cup whose status is to be obtained.
Vac1	Data type: Int variable Current vacuum level of the suction cup's first channel obtained, range 0-100.
Contact1	Data type: Int variable Current status of the suction cup's first channel obtained, range 0-255, where bit6-7 indicates whether the an object is detected. See the table below for status details.

Bit	Name	Value/Description

0	gAct	0: the electric suction cup is not enabled; 1: the electric suction cup is enabled
2	gMode	0: the automatic control mode; 1: the advanced control mode
3	gGTO	0: adjustment stopped; 1: the pressure or vacuum is being adjusted
4-5	gSTA	0: the electric suction cup is not activated; 1 & 2: the electric suction cup is not used; 3: the electric suction cup is activated
6-7	gOBJ	0: below the minimum air pressure; 1: work object detected and minimum pressure value reached; 2: work object detected and maximum pressure value reached; 3: no object detected, object lost or detached.

Time_Err1

Data type: Int variable

Whether the suction cup's first channel obtained triggers a timeout alarm.

Vac2

Data type: Int variable

Current vacuum level of the suction cup's second channel obtained, range 0-100.

Contact2

Data type: Int variable

Current status of the suction cup's second channel obtained, range 0-255, where bit6-7 indicates whether an object is detected. See the table above for status details.

Time_Err2

Data type: Int variable

Whether the suction cup's second channel obtained triggers a timeout alarm.