CBSH Robot

Shenzhen ChipBroad & SkyHub Robot Co., Ltd

CBSH TCP/IP Remote Control Interface Documentation

Shenzhen ChipBroad & SkyHub (CBSH) Robot Co., Ltd

Table of contents

Foreword

Purpose

This manual introduces CBSH Studio Pro robotic arm control software is designed for the CRA/CRAF series, the new version of Nova, and Magician. The E6 robot's functions and operations are explained to help users understand and use CBSH Studio . Pro.

Reader

This manual is applicable to:

- Client
- Sales Engineer
- Installation and commissioning engineer
- Technical Support Engineer

Related documents

Document	illustrate	Download link
CBSH CRA Series Hardware User Manual	Introducing CBSH CRA/CRAF series association This includes the robot 's functions, technical specifications, and installation instructions .	Visit the CBSH Robotics official website and click "Service and Support". > The " Download Center " allows you to search by document name or filter by product series .
CBSH New model Nova Series Hardware UserManual	Introducing CBSH New model Nova series collaboration This includes the robot 's functions, technical specifications, and installation instructions .	
CBSH Magician E6 User Manual	Introducing CBSH Magician E6 Collaboration. The robot 's functions, technical specifications, installation instructions , etc.	

CBSH TCP_IP Secondary Development Interface Documentation	Introduction to secondary development based on TCP/IP protocol. How to use the interface. In addition to the documentation, you can also obtain secondary versions in various languages from GitHub to develop a demo.
CBSH Bus Communication Protocol Document (EtherNetIP)_Pro finet)	Introducing the robot bus communication function The usage way of (EtherNetIP/Profinet) .
CBSH Collaborative Robot Teach Pendant User Manual	This section introduces how to use the teach pendant that is designed to work with collaborative robots .

Time	Version	Revision history
20 25/05/12	V 4 .6.3	1. Added 2.5 Troubleshooting methods for robot connection failures 2. Added 3.2 Security checksum 3. Other content optimizations and image updates 4. Update to CBSH Studio Pro Version 4.6.3.0
20 25/03/07	V 4.6.0	adding security checks and pop-up window appearance
20 24/12/26	V 4.6.0	1. Added debugging and running functions , making debugging more convenient. 2. The newly added trajectory has been restored , making the operation more flexible. 3. Added file migration feature , one-click import and export of configuration files. 4. Added firmware upgrade feature : one-click firmware version upgrade. 5. Other content optimizations and image updates: 6. Updated to CBSH Studio . Pro Version 4.6.0

20 24/07/31	V 4.5.1	Content optimization and related image updates
20 24/03/25	V 4.5.1	Update to CBSH Studio Pro Version 4.5.1
20 23/11/28	V 4.5.0	Update to CBSH Studio Pro Version 4.5.0
20 23/10/13	V 4.4.1	updating Lua commands GetInRegs, GetHoldRegs, and SetHoldRegs
2023/08/10	V 4.4.0	1. Manual Table of Contents Restructuring 2. Update to CBSH Studio Pro Version 4.4.0
20 23/06/02	V 4.1.1	1. Update to CBSH Studio Pro Version 4.1.1 2. Description and graphic style optimization
20 23/05/16	V 4.1.0	Initial release, corresponding to CBSH Studio Pro Version 4.1.0

Symbol Conventions

In this manual, and their meanings are as follows.

Symbol	illustrate
Danger	This indicates a high potential danger that, if not avoided, could result in death or serious injury.
warn	This indicates a moderate or low potential hazard, which, if not avoided, could result in minor personal injury, damage to the robotic arm , or other similar situations.
Notice	This indicates a potential risk; ignoring this text could lead to damage to the robotic arm, data loss, or unpredictable consequences.
illustrate	This indicates additional information to the main text, emphasizing and supplementing it.

1 Start here

1.1 Overview

Welcome to CBSH Studio Pro. CBSH Studio Pro, which is a control software developed by CBSH robots. It features simple and easy-to-use functions , strong practicality, and a concise and easy-to-understand interface, which can help users quickly master the use of CBSH robots.

This manual primarily introduces how to use CBSH Studio . Pro Control the CRA series robots.

CBSH Studio The Pro version supports use on PCs, Android tablets, and CBSH's self-developed teach pendant. The teach pendant version has some unique functions that require the use of teach pendant hardware (such as three-position switches), which are not described in this document. Please refer to the " CBSH Collaborative Robot Teach Pendant User Manual".

1.2 Configuration Requirements

CBSH Studio The configuration

requirements for Pro are as

follows : PC

Configuration items	Minimum configuration	Recommended configuration
Processor	64-bit Intel or AMD processor, SSE Version 4.2 or later, 2.9GHz or higher CPU frequency	
Operating system	• Windows Windows 10 (64-bit) version 1809 • later 11	
RAM	8 GB	16 GB and above
Graphics card	• Supports DirectX 12 • 2 GB of video memory	• Supports DirectX 12 • 4GB of video memory, suitable for 4K and higher resolutions.
Monitor resolution	1440x 900, 100% scaling	1920x1 080 or higher resolution

Harddisk	4 GB available space	• GB of available space • with built-in SSD

illustrate :

- A processor , memory, or graphics card with
- specifications below the minimum can cause
- software lag or crashes. A monitor resolution below
- the minimum can result in incomplete interface display.

Additional hard drive space may be required during the software installation process.

Operating systems with configurations below the minimum may cause software incompatibility and require further evaluation.

Mobile

Customers can request optional tablets from CBSH. If they wish to purchase their own tablets, the configuration requirements are as follows:

Configuration items	Recommended configuration
Processor	4 cores , 2.0GHz or above
Operating system	Android 10 and above
RAM	4 GB
Storage space	32 GB
Display screen	8- inch

1.3 Install software

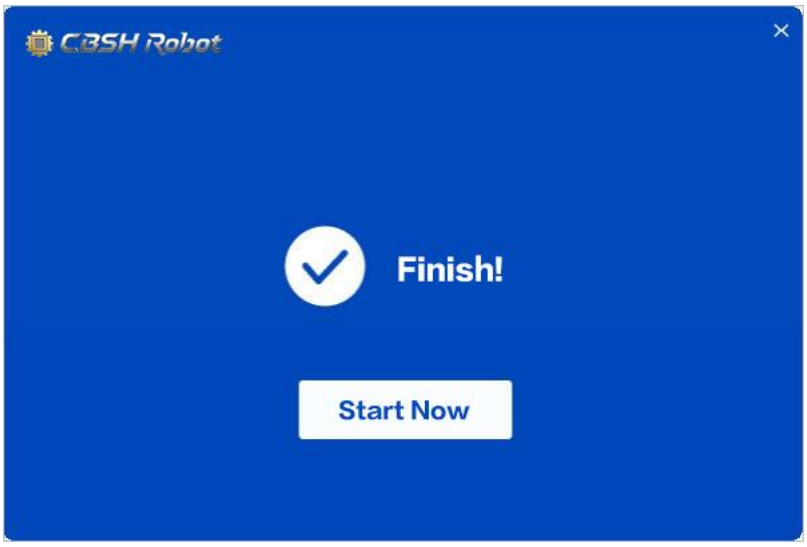
PC end

Download the latest CBSH Studio from the official website . Pro

Install the package and follow the steps below: 1 . After double-clicking to open the installation package, select the installation language and click **Start Installation** .

2. Click " **One-Click Install"** in the installation interface , or set the installation path in the **custom options and then start the installation.**

3. the software is successfully installed, click **"Experience Now"** on the prompt screen to open the software directly.



Mobile

- Android: Please download the latest CBSH Studio Pro from the official website. Install the package and then install it.

1.4 How to use this manual

stage	illustrate	Reference Chapters
Connecting robots	CBSH Studio Pro supports connecting to the robot via wired or wireless means. Most functions require the robot to be connected before they can be used. When the robot is not connected , you can only set the display language or upload mobile crash logs in the system settings.	Power on and connect
Understanding Software	CBSH Studio quickly Pro's main interface	Interface Overview
Understanding Software	And its functions.	Interface Overview
Installation / Voltage Setting	If the robot is not installed on a level surface, the installation angle must be set first . If using DC power... For input , the voltage range also needs to be set.	Install and set the power supply voltage.

Quickly experience the robot function	Write and run a program that allows a robot to move cyclically between two points .	Quick Start
Security Settings	you begin using the robot, please configure its safety features based on the results of the risk assessment .	Security I/O security settings
Understand the basic operation of robots	Learn the basic operations of the robot , including user login, jogging, and alarm handling .	Basic Robot Operation
Programming	Users can control the robot to run automatically by writing programs . First, please choose a suitable programming method and learn how to use the programming interface.	application
	Understand the specific programming instructions, including their functions and usage.	Appendix B Block Programming Block Instructions Appendix C Script Programming Function Description Appendix E Python programming function descriptions
	Users can view and modify the real-time status of IO and debug IO - related functions.	Control cabinet DI/DO control cabinet AI/AO terminal I/O
Use eco-friendly accessories	CBSH + plugins help users quickly configure and use CBSH ecosystem accessories, eliminating the need for secondary development . For information on ecosystem accessories already supported by CBSH, please visit the CBSH official website .	CBSH +

Software and robot settings	Modify the software and robot-related settings. Robot- related settings are stored in the controller, and different robots require separate settings.	set up
Remote control robot	Users can remotely control the robot via the control cabinet's I/O, including controlling the operation of the project and obtaining the robot's status.	Control cabinet DI/DO
	Users can remotely control robots via Modbus (Modbus-TCP or RTU -over-TCP), including controlling engineering operations and acquiring robot status and real-time feedback data.	Modbus Appendix A Modbus register definition
View and export logs	Logs can help troubleshoot problems; if necessary, logs can be exported and sent to technical support for further investigation.	log
Firmware upgrade	the robot firmware to the latest version with one click, or revert to the previous firmware version.	Firmware upgrade

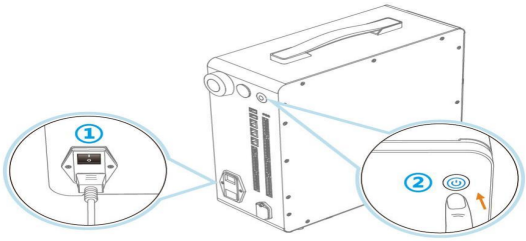
2 Power on and connect

2.1 Power on

CRA / CRAF series

After completing the installation and wiring of the control cabinet and turning on the external power supply, press the switch above the power interface to the " | " position. Then press the round button on top of the control cabinet briefly. When the blue indicator light stays on, it means that the control cabinet has been started.

24

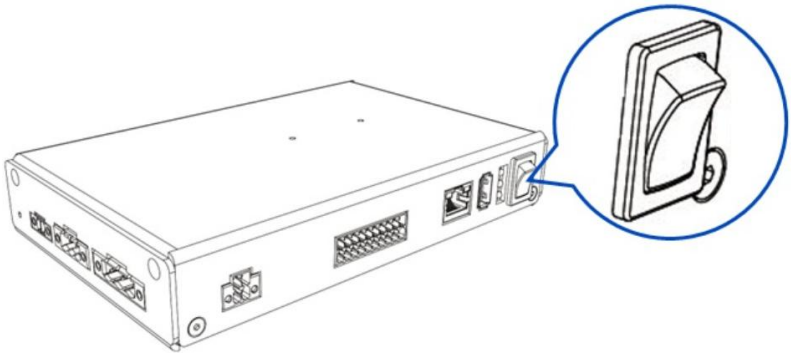


illustrate :

The control cabinet startup process, the orange light on the LAN2 port will flash, indicating that the built-in WiFi router is initializing, and the LAN, WiFi, and teach pendant cannot connect to the robot. Throughout the initialization process, the light will flash faster and faster until it eventually goes out. The screen is off , indicating that initialization is complete. At this point, the user can try to connect to the robot.

New Nova series

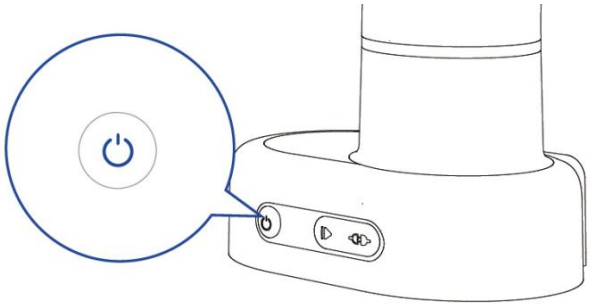
Completing the robot installation and wiring and turning on the external power supply, flip t he rocker switch on the control cabinet upwards. When the indicator light is constantly blue, it means that the control cabinet has been started.



Magician E6

Completing the robot installation and wiring and turning on the external power, briefly

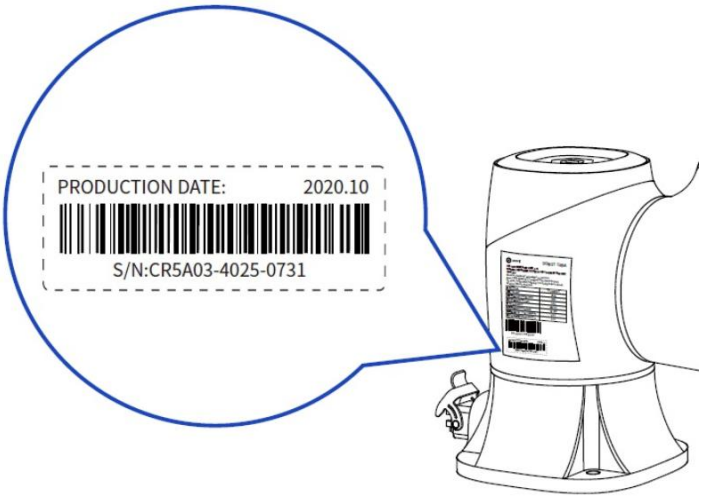
press the round button on the robot base. The robot's indicator light will remain solid blue, indicating that the robot has started.



2.2 Wireless connection

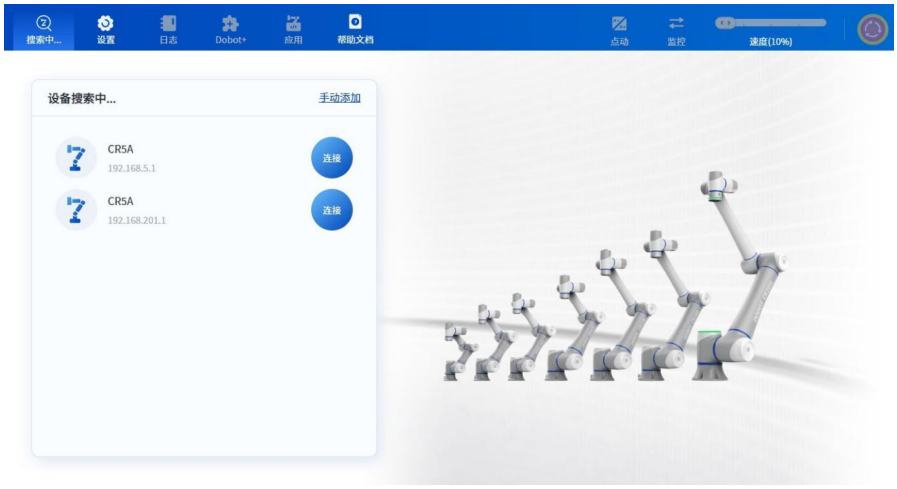
> illustrate :
> For wireless connection to Magician For the E6 robot, please purchase the wireless module separately and plug it into the USB port on the robot base.

on your PC or tablet . The default SSID for the WiFi is " CBSH Product Model - Serial Number", where the serial number is two 4-digit numbers connected by "-" in the robot's S/N (the S/N code is affixed to the nameplate on the robot arm base; for example, the default SSID for the CR5A shown in the image below is " CBSH CR5A-4025-0731", and other models are similar). Initially... The Wi -Fi password is: 1234567890. The Wi-Fi SSID and password can be changed in the communication settings .



connected via Wi -Fi, the robot's IP address is 192.168.201.1.

PC end



Open CBSH Studio In Pro mode, the software will automatically search for connectable robots and display the search results on the interface. Because... Due to the internal network architecture of the control cabinet, CBSH Studio The Pro may find multiple different IPs for the robot. You can connect to the robot normally through any of them . It is recommended to use 192.168.201.1 to connect.

Click the robot to the right 

of the one you want to connect to.

Mobile

Click the top left corner 

of the interface

illustrate :

the software connects to the robot, it checks whether

- the software version and the controller version
- match: if the first two digits of the version number match, the robot connects normally.
  If the first two digits of the version number do not match, the software will display a pop-up message indicating a version mismatch and automatically disconnect.

Successfully connecting to the robot, the software interface will refresh, displaying relevant

information and a 3D model of the connected robot. Clicking the **disconnect button** to the rig ht of the robot's name will disconnect the current robot.

2.3 Wired connection

Only on PC.

The network cable to the LAN port of the control cabinet and the other end to your PC. Then, change the PC's IP address to be on the same network segment as the control cabinet 's IP address . The default IP address for LAN1 is 192.168.5.1, and the default IP address for LAN2 is 192.168.200.1. The IP address of LAN1 can be modified in the communication settings , while the IP address of LAN2 cannot be modified.

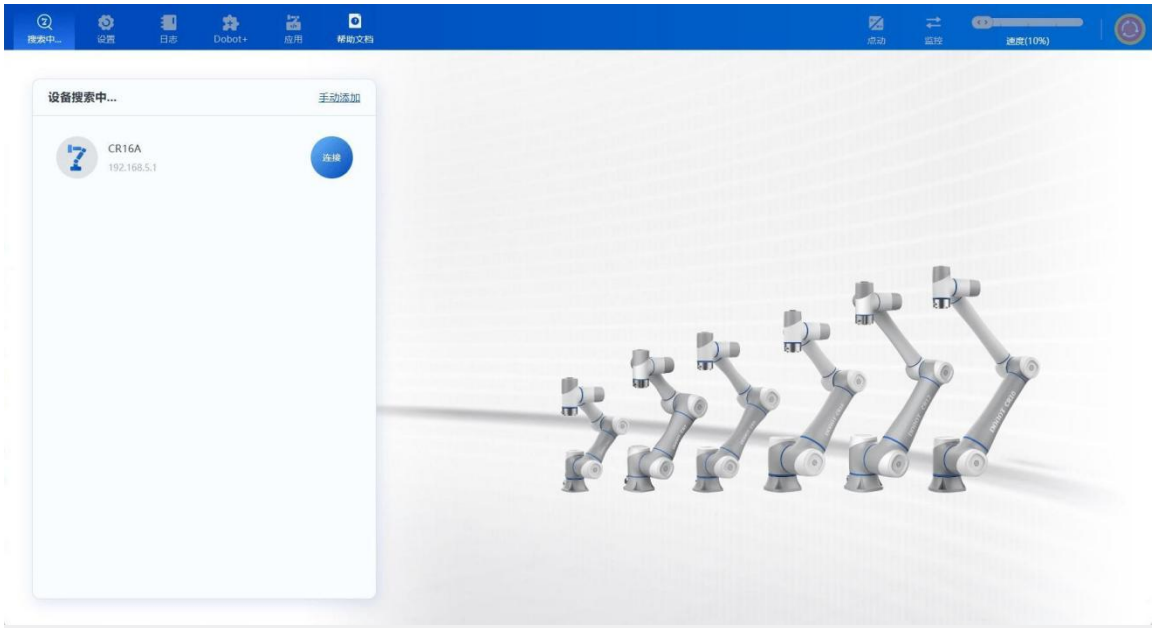
The methods for changing a PC's IP address vary slightly across different versions of

Windows. This article uses Windows as an example. Let's take 10 as an example.

1. in the taskbar and select **to view network connections** .
2. the current network connection (such as **Ethernet**) and select **Properties** . Then, in the pop-up window, find and double-click **Internet Protocol Version 4 (TCP/IPv4)** .
3. In **the Internet Protocol Version 4 (TCP/IPv4) properties** interface, select " **Use the following IP address"** and modify the PC's IP address, subnet mask , and default gateway. You can change the PC's IP address to any unused IP address within the same network segment as the control cabinet. The IP address , subnet mask , and default gateway must match those of the control cabinet. For example, if the computer's IP address is set to 192.168.5.100, the subnet mask...
   It is 2 55.255.255.0.

4. Click the right side of the robot 

you want to connect to.



illustrate :

The software connects to the robot, it checks whether the software version and the control cabinet version match:

- If the first two digits of the version number match, the robot can be connected normally.
- If the first two digits of the version number do not match, the software will display a pop-up message indicating a version mismatch and automatically disconnect.

5. Successfully connecting to the robot, the software interface will refresh, displaying relevant information and a 3D model of the connected robot. Clicking the **disconnect button** to the right of the robot's name will disconnect the current robot.

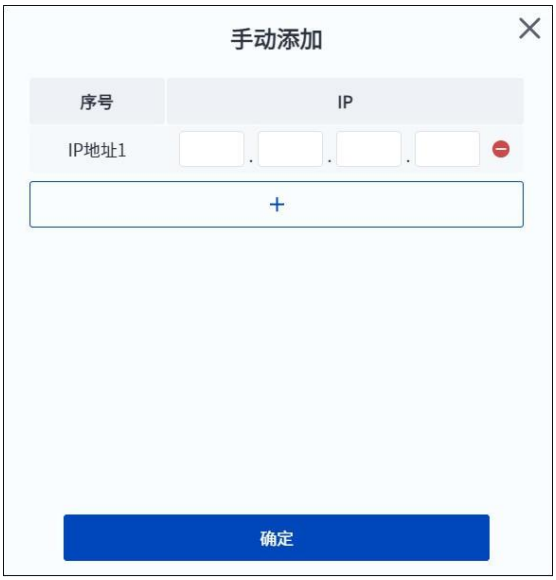
illustrate :

To connect multiple robots to a single computer, you can connect the robots and th e computer to the same local area network, and then open multiple CBSH Studio in stances on the computer. In the Pro window, connect to different robots.

2.4 Add manually

If no robot is found, you can manually add the robot's IP address to search.

Open CBSH Studio After selecting Pro, click " **Add Manually** ". In the pop-up window, click... + Add robot IP.



A maximum of 5 robot IPs at a time . After entering the IP addresses, click **OK** . The search wi ll prioritize manually added robots.

2.5 Troubleshooting methods for robot connection failures

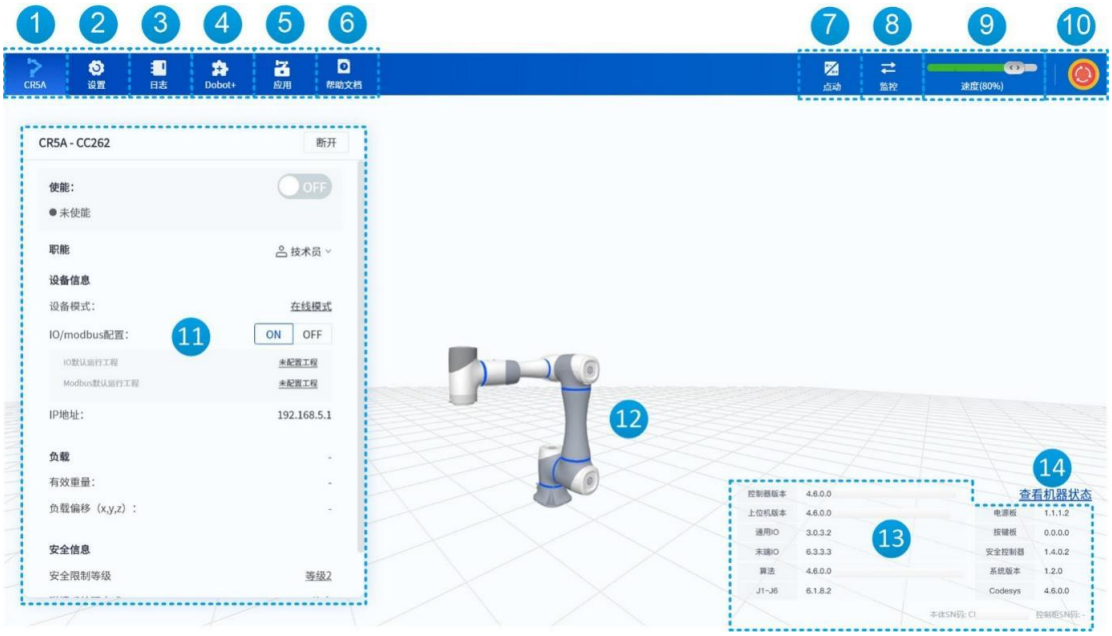
If the robot and CBSH Studio If the Pro connection fails, please check the following ta ble first; if the problem persists after checking , please contact technical support pro mptly.

step	operate	illustrate
1	Check network cable connection	Normal phenomenon: The indicator light on the control c abinet network port is flashing. If there is an issue , please replace the network cable for testing or check the router's power supply.
2	Disable VPN/Proxy	Exit all VPNs/proxies, and restart your computer and control cabinet.
3	Perform a ping test (e.g.: ping 192.168.5.1)	Normal phenomenon: A data packet is received in response. If there is no response, please check if the control cabinet and computer IPs are on the same network segment. See the wired connection section for instructions on changing the IP address .
4	Restart the control cabinet	blue indicator light means the control cabinet has been started.
5	Check software and firmware versions	Control cabinet firmware and CBSH Studio The Pro software version must be compatible.
6	Check wired conne ction configuration	in the taskbar and select **to view network connections** . the current network connection (e.g., **Ethernet**) and select **Properties** . Then, in the pop-up window, find and double-click **Internet Protocol Version. 4 (TCP/IPv4)** . In **the Internet Protocol Version 4 (TCP/IPv4) properties** interface, select " **Use the following IP address** " and confirm that the PC's IP address and th e control cabinet's IP address are on the same networ k segment.
7	Check wireless connection configuration	in the taskbar and select **to view network connections** . the current network connection (such as **WLAN**) and select **Properties** . Then, in the pop-up window, find and double-click **Internet Protocol Version 2. 4 (TCP/IPv4)** . In **the Internet Protocol Version 4 (TCP/IPv4) properties** interface, select " Obtain **an IP address automatically "** .
8	Check permissions and firewall	Temporarily disable the firewall and security software. Run the software as administrator .

3 Interface Overview

3.1 Main interface

The software connects to the robot, the main interface displayed is shown in the figure below.



Serial Number	illustrate
1	Click to open the main interface (current interface). The color of the robotic arm icon is synchronized with the indicator lights on the robot, changing according to the robot's status. For details on the in dicator light color definitions, please refer to the corresponding hard ware user manual for the robot.
2	Click to open the settings page , where you can configure parameters related to the robot's installation, movement, and safety.
3	Click to open the log page and view alarms and logs.
4	Click to open the CBSH + page , where you can install and use the CBS H + ecosystem plugins.
5	Click to open the application page , where you can program using blocks or scripts.
6	Click to open the help document, which can be displayed in the software interface, in a separate window, or in a browser.
7	Click to open the jog panel and control the robot to jog or incline.

8	Click to open the monitoring panel , which allows you to monitor the robot's I/O and global variables, etc.
9	Used to display and set the global rate, controlling the speed of the robotic arm's movement.
1 0	Emergency stop button : Press this button during the operation of the robotic arm to bring it to an emergency stop in case of an emergency.
1 1	This panel is used to enable the robot, switch user roles, set device information, and display load information and safety information (such as security checksum information).
1 2	The robot's 3D model maintains the same posture as the real robot.
1 3	Robot serial number (SN) and firmware version information. contacting technical support to report an issue, please send a screenshot of this information to them so they can locate the problem. If you need to upgrade the firmware, please use the robot maintenance tool first; or upgrade via firmware upgrade .
1 4	Click to view the robot's current voltage, current, temperature, and other statuses. 

3.2 Security checksum

Overview

A security checksum represents the result of calculating parameters within a check range using a specific check algorithm. If the parameters within the check range change, CBSH Studio ... **The verification and display content** on the Pro homepage will also change synchronously, and user logs will record security information. Modification of the full checksum.

Operations that cause checksum changes

Via CBSH Studio Modifying security-related parameters in the Pro software (as shown in the table below) will update the verification code on the homepage.

CBSHStudio Pro	Operation
Motion parameters (CRA)	Modify the torque constraint switch, such as changing it from OFF to ON, to adjust the reproduction speed.
Motion parameters (Magician) E6)	Modify teaching speed and playback speed
Installation settings	Modify the installation posture, such as changing from horizontal to suspended, wall-mounted downwards, or wall-mounted upwards; modify the tilt angle and rotation angle of the custom installation posture.
Security Restrictions (CRA)	Modify TCP force, power, TCP velocity, momentum, stopping time, stopping distance, and modify the post-collision handling method. Modify rollback distance
Collision detection (Magician) E6)	Modify the collision post-processing method and adjust the backoff distance.
Safety wall	Modifying any parameter in the security wall interface will trigger a checksum update.
safe zone	Modifying any parameter in the secure area interface will trigger a checksum update.
Safety origin	Modify the joint angle of the safety origin
Mode settings	Modify manual/automatic mode, such as changing from manual/automatic mode (never enabled) to manual or automatic mode.
Safe I/O	Modify the function configuration of SI and SO

illustrate :
Modifying the enable state of the three-position switch via the teach pendant will also cause the homepage verification code to update.

Performing the following operations will cause the security verification and detection to

fail; a corresponding alarm message will pop up after successfully connecting to the robot. Users must resolve the security verification and mismatch issue according to the actual situation before they can continue to use the robot.

operate	CBSH Studio Pro pop-up window
(1) Manually modify the security verification intermediate file (2) Manually delete the security verification intermediate file (3) From CBSH Studio Upgrade your Pro version from below 4.6.0 to 4.6.0 or higher . (4) From CBSH Studio Pro Downgrade from version 4.6.0 and above to a version lower than 4.6.0 .	
(5) From CBSH Studio Pro Version 4.6.0 has been upgraded to version 4.6.0 or later , and the checksum verification range has changed. (6) Bypass CBSH Studio Pro uses other tools to modify security parameters, such as manually modifying security parameters in configuration files.	

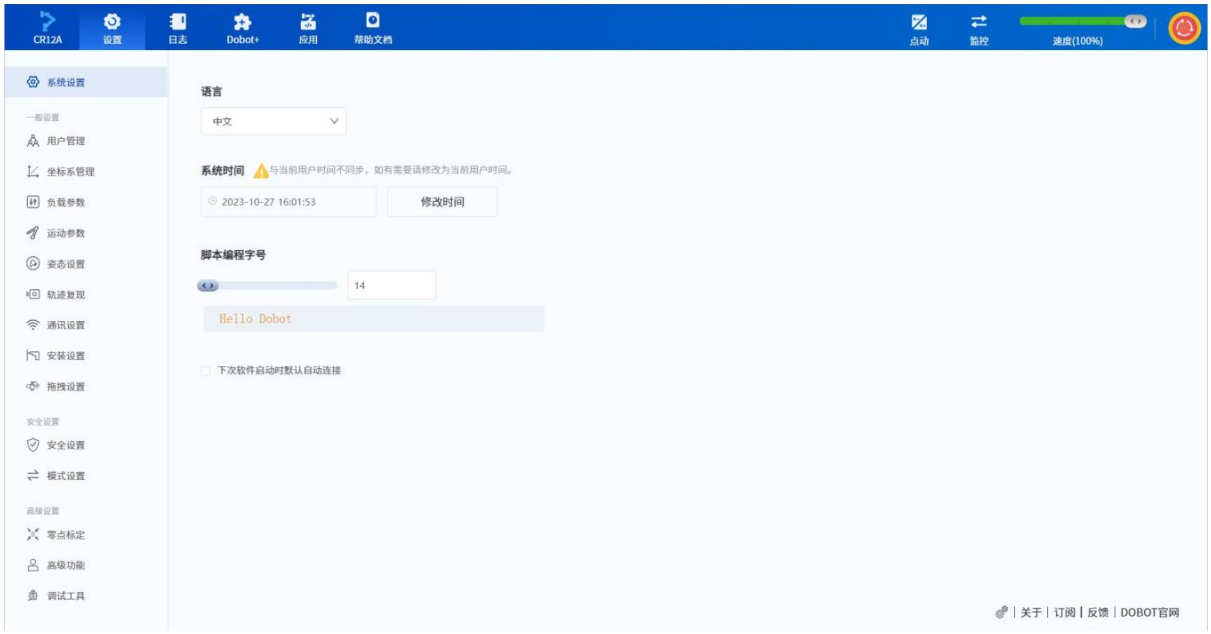
- Click " **Use current security parameters** " to generate a new verification code based on the current system's security parameters. A message will appear indicating **successful setting of the current security parameters** .
- Click **"Restore Security Parameters"** to restore the previously correctly configured security parameters. The verification code on the homepage will remain unchanged. A message will appear indicating **successful restoration of default security parameters** .

illustrate :
CBSH Studio is involved in the subsequent events The operation that triggers the verification code update may vary due to changes in the Pro interface position or interface parameters . Please refer to the actual product for details.

3.3 Settings page

The settings page allows users to modify software and robot-related settings, such as display language, user/tool coordinate system, security settings , etc. See the user

manual for each settings page for details .



3.4 Log page

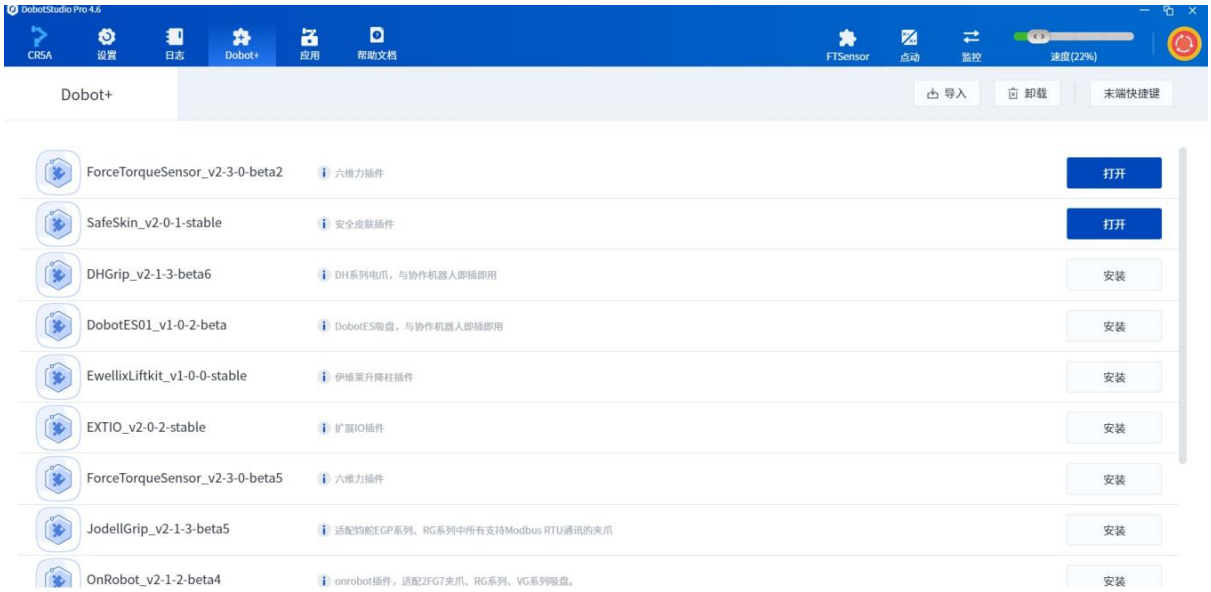
The log page includes two subpages: **the current alarm** and **the log records .**

- **The current alarm** page is used to view and clear alarms. See "Handling Alarms" for details .
- **The log** page is used to view and export logs; see the logs for details .

3.5 CBSH + Page

The CBSH + page is used to manage and use CBSH + plugins. See the CBSH + feature description for details.

CBSH + plugin is a dedicated plugin developed by CBSH for ecosystem accessories. Users can directly configure and use ecosystem accessories such as grippers through the plugin without secondary development.



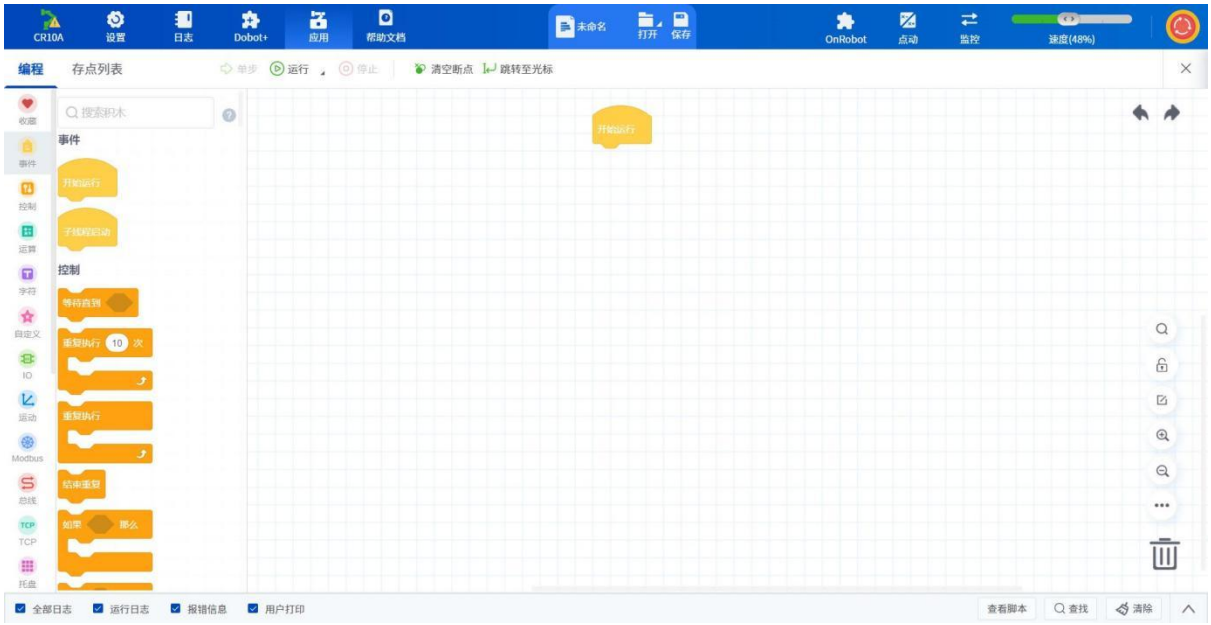
插件及手册下载

3.6 Application Page

The application interface allows users to create and run their own projects, enabling robots to operate automatically.

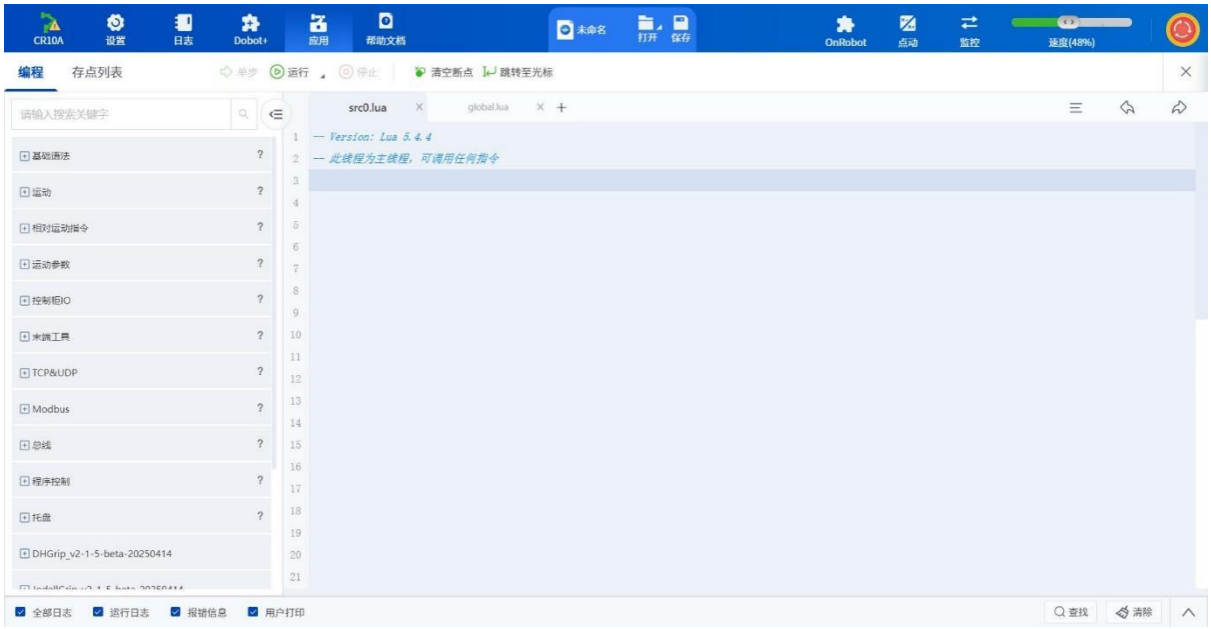
Block Programming

Block programming is a graphical programming method where users can drag blocks from the left side of the programming page to the right canvas to program. See Block Programming for details .



Scripting

Script programming is a programming method based on the Lua programming language. Users can find the commands they want to use on the left side of the programming page, configure the parameters and add them to the right programming area, or write programs directly in the right programming area. See Script Programming for details .



Python programming

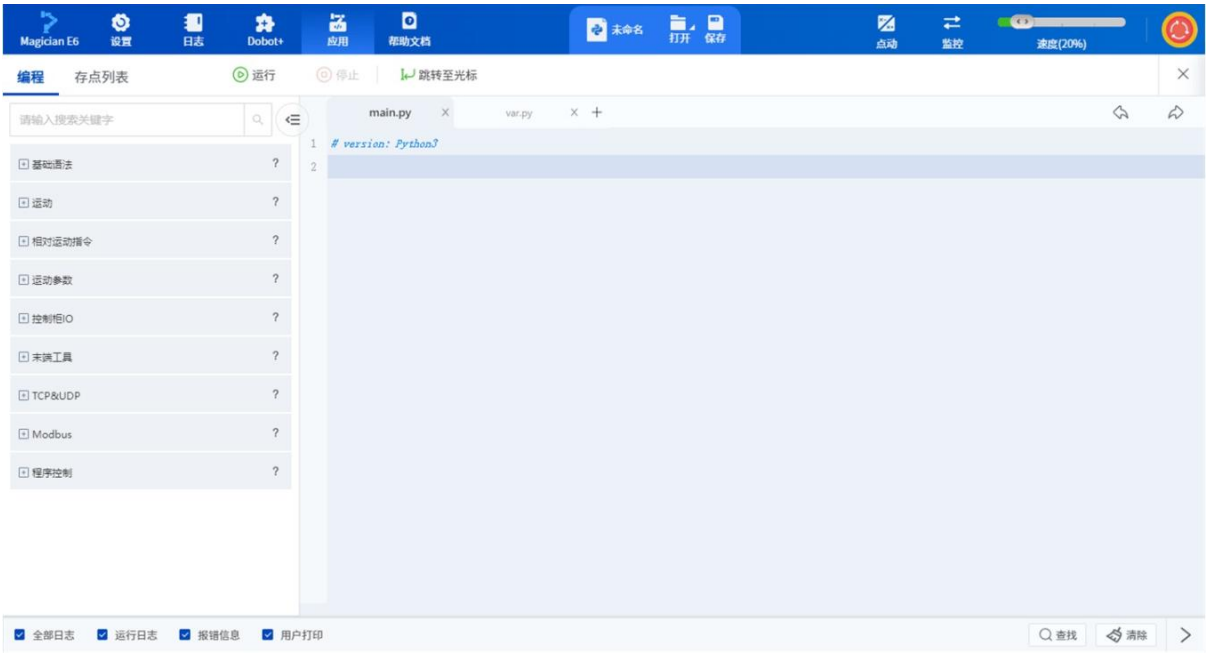
Python programming is a programming method based on the Python programming

language, and **it only connects to Magician on the PC side. This software is compatible with the E6 robot** and is used for teaching and research. The user interface is

basically

the same as that of script programming, and debugging is not supported. See Python

programming for details .



3.7 Jog panel

The jog panel is used to control the robot to perform jog or inching movements, supporting both joint jog and coordinate system jog modes. See the jog function description for details .

Click the top right corner of the panel The panel can be turned into a free-draggable independent window; after closing the independent window and reopening the click page , the click page will turn back into the embedded panel.

3.8 monitoring panel

The monitoring panel allows you to monitor and configure the status and function of each robot's I/O, and also manage and view global variables. See the user manuals for each monitoring page for details .

Click the top right corner of the panel. The panel can be turned into a free-draggablein dependent window; after closing the independent window and reopening the monitorin g page , the monitoring page will revert to the embedded panel.

4 Quick Start

This chapter will introduce how to create a modular project that shows a robot moving cyclically between two points, helping users quickly experience the functions of the CBSH robot.

This chapter only introduces the simplest operating steps to achieve the goal. For detailed explanations of each function, please refer to the subsequent chapters of this manual.

Enabling robots

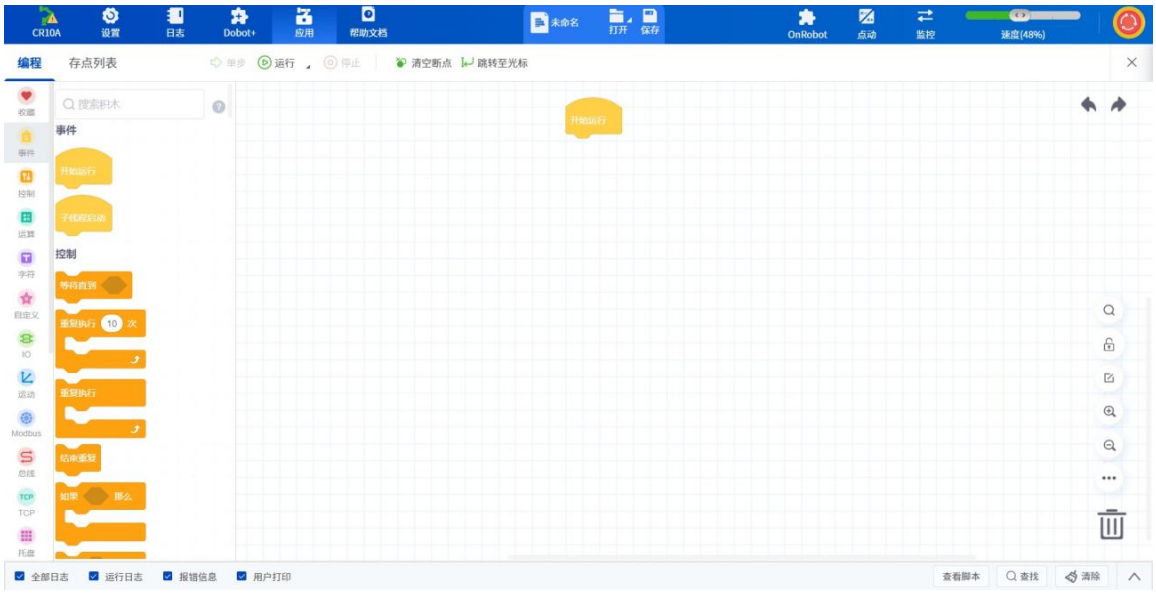
1. Successfully connecting to the robot, click **the enable switch** on the main interface
2. Information panel .

2. The software will display **a load enable settings** window. If the tool is not installed on the robot end effector, simply click the "OK" button **to enable** . If the tool is already installed on the robot end effector, please refer to the enable page to set the load parameters.

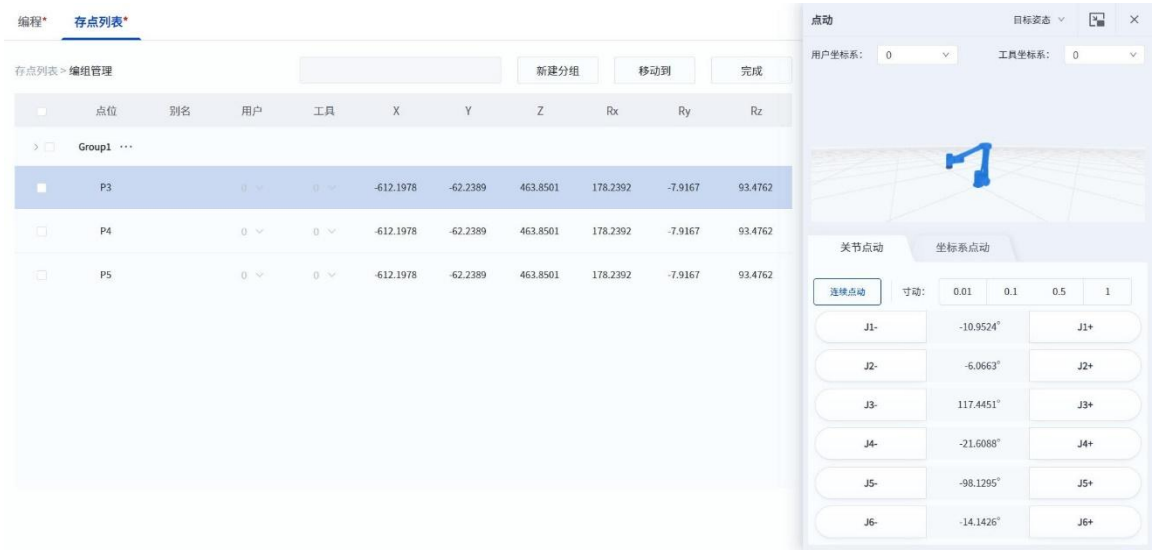
3. After successful enabling, the enable button will turn **ON** .

Creating a block project

1. Open **the application > Block-based programming** interface.



2. Open **the save point list** .



3. Press and hold the control buttons on the **jog panel (e.g., J1+ , J1-**) to move the
4. Robotic arm to the desired pose, then click the **"Add Point" button** in the upper right corner of **the save point list** to save point P1.
4. Similarly , save point P2.
5. Return to **the programming page, and use the block group** on the left to drag **the**
6. **Repeat execution block below the start execution** block on the canvas .
6. Drag **the Motion to Point** block from the **Motion Blocks group to the Repeat Execution** block, click the Point drop-down box, and select P1.
7. Drag another **Motion to Point** block below the previous **Motion to Point** block, click the Point dropdown menu, and select P2. The final result is shown in the image Below

Save and run

> ⚠ Notice :
> Starting the robot, make sure there are no people or any other obstacles within the robot's working range.

Click the top of the programming page Save , enter a project name (e.g., "test") and save the current project, then click Save. During operation , the robot will first move t o point P1, and then begin to cycle between points P1 and P2.

5 Basic Robot Operation

5.1 User Login

Users can assign different functions to different users of the robot to achieve operation permission management.

CBSH Studio After Pro connects to the robot, it will automatically log in to the default function (which can be modified). If the default function has no password, the user will not notice anything ; otherwise, a login window will pop up asking for a password. If you wish to log in to a different function, you can also switch functions in this window.

CBSH Studio Pro supports four roles, each with different operating permissions (which can b e modified).

- Administrator : Has full access to all functions by default. Default password: 888888.
- Technician : The default login function for the robot when it leaves the factory. By default, they have access to all functions except for advanced settings (installation and security related settings) . No password is required by default.
- Operators : By default, they only have basic robot operation permissions such as jogging and engineering operation. There is no password by default.
- Custom functions: Functions created by the administrator with customizable names and permissions; they are not displayed if not created.

the current user does not have permission to operate (such as **edit** buttons) will be grayed out. Clicking them will display a message indicating that the user does not have permission to operate, as shown in the image below.

To switch to a function with operating permissions, please return to the main interface and switch functions in the information panel. Switching to a function requiring a password will require entering the password; switching to a function without a password will be done directly.

When a user logs in as an administrator, they can set the default login role, manage custom roles, and view/modify the permissions and passwords for each role. See User Management for details .

5.2 Enable

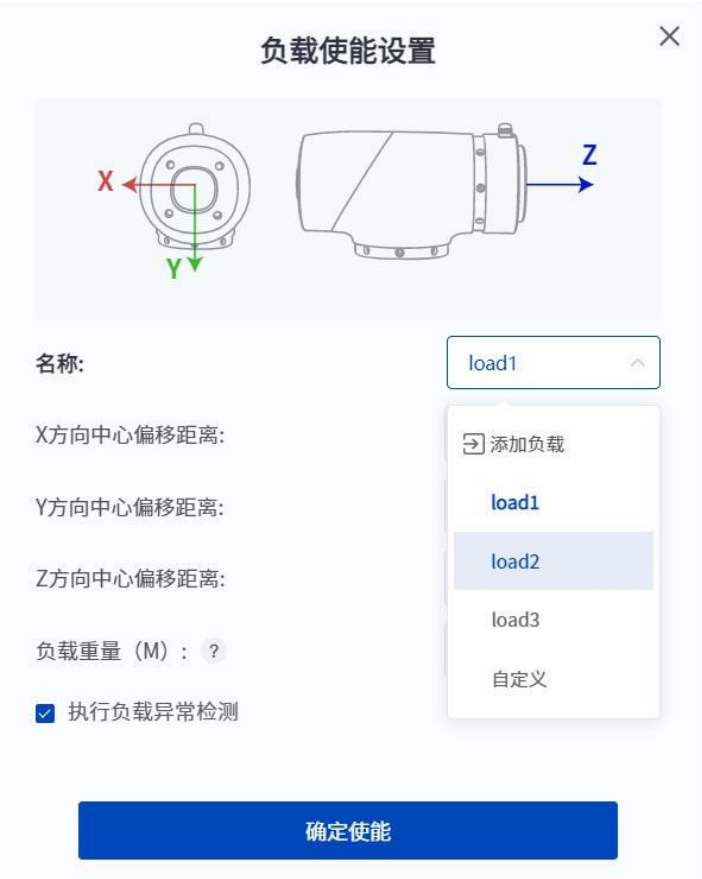
The robot is powered on, it is disabled by default. You can configure and program the robot in this state, but you cannot control its movement or program execution . To control the robot's movement or program execution, you need to enable the robot.

Users can switch the enable state via the enable switch on the main software interface or the enable button on the robot.

Enabled by software

The software enable switch is located on the information panel of the main interface.

The enable button is **OFF** , clicking it will bring up the load settings window.



The **name** dropdown menu allows you to select existing load parameters, **customize them** , or **add new loads** . When you select **custom** , you can manually modify the following parameters:

- **X/Y/Z direction center offset distance** : the offset distance of the end load centroid in each direction. See the schematic diagram on the interface for each coordinate axis direction.
- **Load weight** : The sum of the end effector weight and the workpiece weight must not exceed the robot's maximum allowable load.

> ⚠ Notice :
> When connecting a CR20A/CR20V robot, enabling **the Restricted Overload Mode switch in** the Advanced Settings page with administrator privileges will expand **the load weight** setting range to 0. ~ 25kg.

- **Perform load anomaly detection** : When this option is checked, the robot will move slightly to detect if the difference between the user-set load parameters and the actual load is too large. If the difference is too large, the enable will fail, and the load

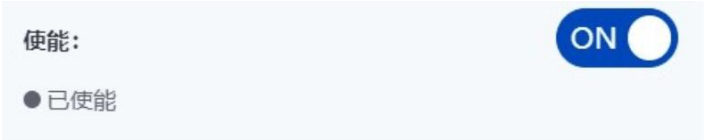
parameters need to be reset or this option needs to be unchecked.

> ⚠ Notice :
> - It is recommended to check **the "Perform load anomaly detection "** option
> - when enabling the robot to ensure operational safety and equipment protection. When connecting a CR20A/CR20V robot, **the "Execute Load Anomaly Detection" option** should always be checked and should not be modified. Clicking the checkbox will prompt that **load anomaly detection must be forced when the limiting overload mode is enabled** .

"Add Load" from **the name** dropdown menu , you will be automatically redirected to **settings. >** Load parameters **> Add** a page and add new load parameters .

> ⚠ Notice :
> If there are already 10 loads on the load parameters page, clicking **"Add Load"** will prompt that **a maximum of 10 load parameters can be configured** .

After successful enabling , the enable button will turn **ON** . Clicking it again will enable the next step.

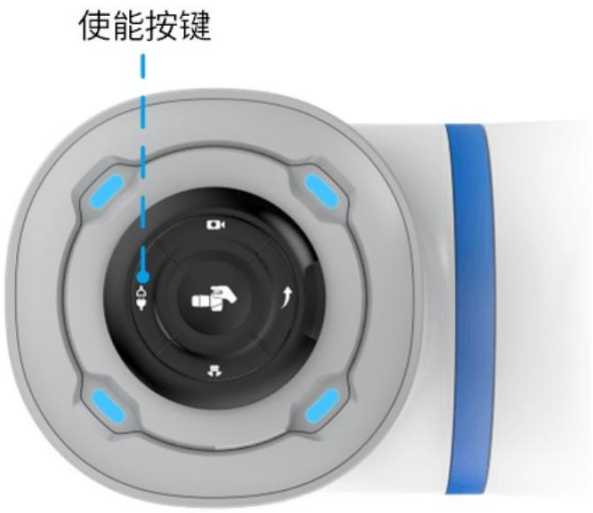


Enable via robot button

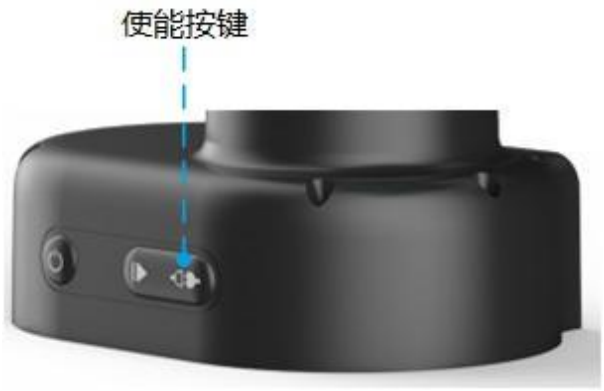
Robot enable button location

- The enable button for the CRA/CRAF series (excluding CR20A/CR20AF) robots is located at the end of the robot.

> ℹ illustrate :
> The CR20A /CR20AF/new version Nova robot has no enable button and can only be enabled via software.



- Magician The enable button for the E6 robot is located on the robot's base.



Robot Enable Button Operation

- Enable operation : When the robot is powered on but not enabled (blue indicator ligh t is always on), press and hold the enable button for about 1.5 seconds until the purp le indicator light flashes, then release the button to enter the enabled state (the indic ator light turns green and stays on).
- To enable the function: Press and hold the enable button for about 1.5 seconds until the purple indicator light flashes, then release the button to exit the enable state.

> **illustrate :**
> The robot via the button, the robot will use the load settings from the last time it was e nabled. If this is the first time enabling the robot , the load parameter values will all be 0.

5.3 Recovery Mode

When the robot is enabled in an unsafe area, the pop-up window shown below will appea r, and the indicator light at the robot's end will turn orange, indicating to the user that it has entered recovery mode.

Recovery mode , users can only control the robot's movement by **jogging** or **dragging** .

Once the robot enters a safe area, it will automatically exit recovery mode and display a pop-up notification.

5.4 Device mode

Device mode indicates the robot's current control mode:

- **Online mode** is the default control mode, in which CBSH Studio can be used. The Pro controller allows for robot control and can also be used for remote robot control via I/O or Modbus.
- **TCP mode** is only used when the customer develops their own control software based on TCP. Except for emergency stops, CBSH Studio... The Pro software will not be able to operate the robot; it can only view the robot's status and related settings. If you need to develop your own control software, please contact us. Technical support is available for TCP_IP Remote Control Interface Documentation (V4)

Users can view the current device mode on the information panel of the main interface, and click on the underlined text to switch device modes.

illustrate :

- When the robot is in manual/automatic mode , the device mode cannot be switched. You need to first turn off manual/automatic mode through t
- he mode settings .
  When the robot is running or paused, the device mode cannot be switched.
- the robot switches to TCP mode, if you click the "Disable Operation" button, a message will appear indicating **that the machine is currently in TCP mode and cannot be operated!**

5.5 IO/Modbus Configuration

The IO/ Modbus configuration switch controls whether remote IO/Modbus inputs are enabled. For safety reasons, it is generally recommended that robots... Controlled by only one input source, therefore when the user uses CBSH Studio When using Pro to control the robot, it is recommended to turn off the IO/Modbus configuration switch. Click **the IO/Modbus configuration switch** on the main interface information panel to toggle it ON/OFF.

⚠ Notice :
Configure the I/O and Modbus runtime project through I/ O settings and Modbus settings .

The specific effective range of I/ O/Modebus input is also affected by manual/automatic mode , as detailed in the table below.

Operating mode	IO/ Modbus Configuration ON	IO /Modbus configuration OFF
Manual/ automatic mode not enabled	All IO/Modbus inputs are valid.	All IO/Modbus inputs are invalid.
Automatic mode	Automatic mode has valid IO/M odbus inputs.	All IO/Modbus inputs are invalid.
Manual mode	Manual mode allows for the use of valid IO/Modbus inputs.	All IO/Modbus inputs are invalid.

Remote IO/Modbus Control

Function	Manual mode	Automatic mode
start	X	√
stop	X	√
pause	X	√
Enable	√	√
Enable	√	√
Clear alarm	√	√
Enter drag and drop	√	X
Exit drag and drop	√	X
Select alternative projects	X	√

illustrate :
manual mode, dragging cannot be initiated if a pause is triggered by a collision.

Safe I/O

Function	Manual mode	Automatic mode
User emergency stop	√	√
Protective stop	X	√
Protective stop reset	X	√
Reduced mode	X	√

illustrate :

- When manual/automatic mode is not activated , or after a protective stop is triggered in manual mode.
- However, running the project and reproducing the trajectory are not allowed . automatic mode, all operations are prohibited, including jogging and dragging the robot, running programs, and reproducing trajectories.

5.6 Manual/Automatic Mode

CBSH Studio Pro supports enhancing the security of field applications by setting different operating modes, which can be enabled in the mode settings .

illustrate :
The CR20 A robot is set to **manual mode by default** , while other models are set to n ot have manual/automatic mode enabled by default.

- **Manual mode** : This mode is generally used for programming and debugging the robot.
- **Automatic mode** : This mode is generally used for automatic operation after the robot goes online.

Enabling **the operation mode switching function** in the mode settings , a **manual/automatic mode switch** will appear on the main interface information panel , with **manual** mode as the default.

You can click the switch to switch to **automatic** mode. If an automatic mode password is set, you need to enter the password before you can switch.

When the robot is running, it cannot switch between manual and automatic modes; when the robot is paused or stopped, it can switch between manual and automatic modes.

The permitted operations in manual /automatic mode can be found in the IO/Modbus configuration .

5.7 Inching

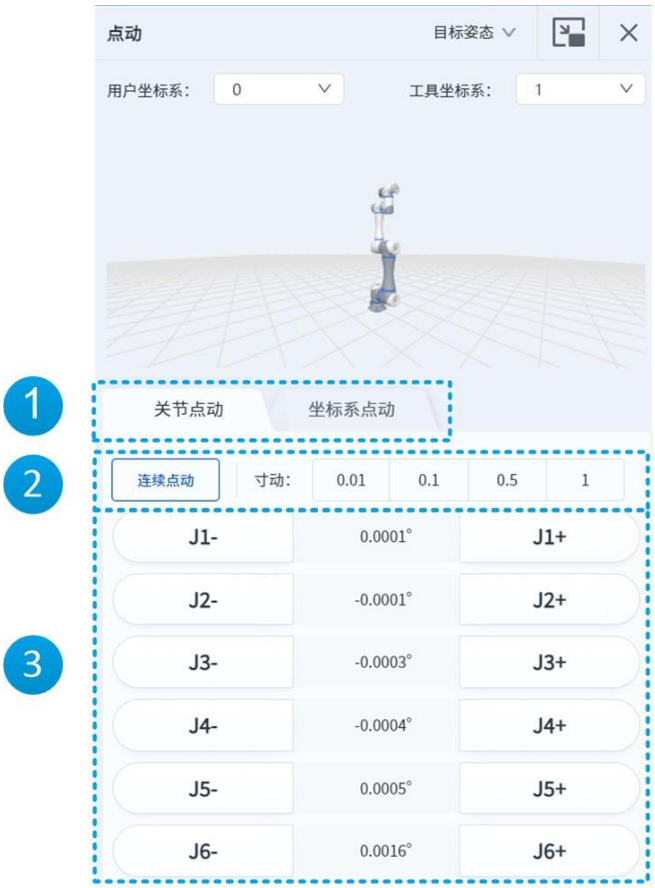
Users can access CBSH Studio The Pro mode allows for manual control of robot movement

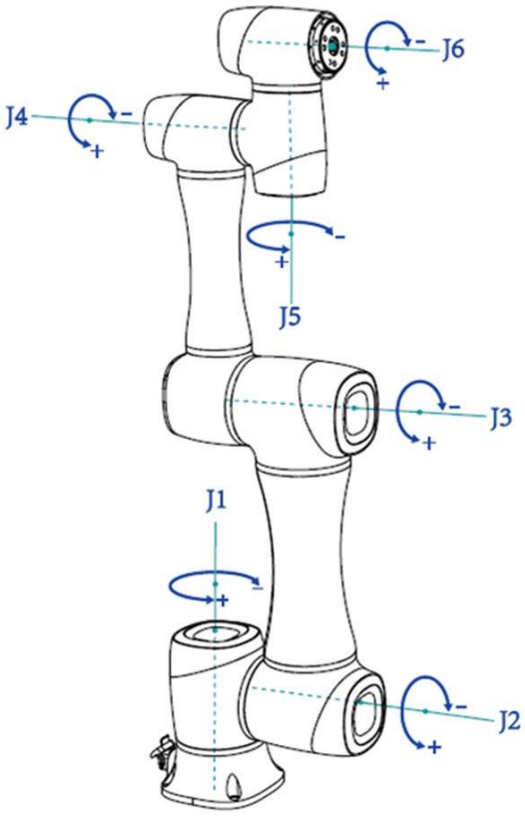
and is typically used for teaching points.

To use the jog function, click on the top toolbar. 

The coordinate system can be continuously moved in a point or inching motion.

Joint movement





Joint kinetics refers to controlling the rotational movement of a single joint of a robot.

To perform joint jogging, first click the **joint jogging** label in area ① of the image above, a nd then control the movement of a single joint using the motion buttons in area ③ .

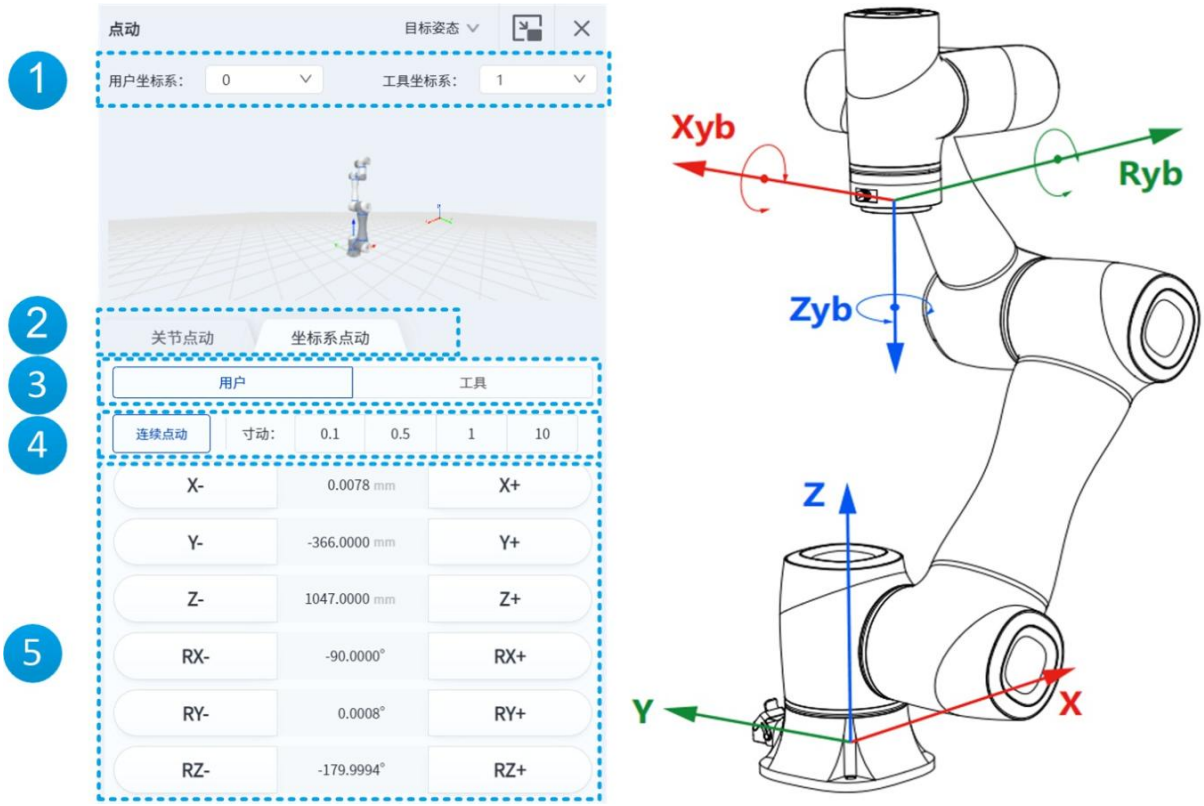
Please refer to the diagram on the right side of the image above for the position and

direction of rotation of each joint axis.

The options in area ② are used to set how the motion buttons are activated:

- Continuous inching: Press and hold the motion button to make the robot move continuously, and release the button to stop the robot from moving.
- Inching: Each click of the motion button will move the robot a specified inching distance (unit: °), and a long press of the button will only trigger one inching motion. teaching the robot to the target location, you can first move the robot to the vicinity of t he target location by continuous jogging, and then make fine adjustments by inching.

Coordinate system jog



Coordinate system jogging refers to controlling the robot's TCP (Tool) Center The Point (the origin of the current tool's coordinate system) is translated and rotated along the specified coordinate system .

To perform coordinate system jogging, first click the **coordinate system jogging label in area ② of the image above, and then control the movement of** the T CP using the motion buttons in area ⑤ . The diagram on the right side of the image above uses the user coordinate system 0 as an example to show the definition of each coordinate axis and its rotation direction.

The options in area ④ are used to set how the motion buttons are activated:

- Continuous inching: Press and hold the motion button to make the robot move continuously, and release the button to stop the robot from moving.
- Inching: Each click of the motion button will move the robot a specified inching distance (X/Y/Z unit: mm, RX/RY/RZ unit : °). A long press of the button will only t rigger one inching motion.

Teaching the robot to the target location, you can first move the robot to the vicinity of t he target location by continuous jogging, and then make fine adjustments by inching.

Switching motion reference coordinate system

Users can modify the robot's current user coordinate system and tool coordinate system in area ①. Users can also configure **settings .** > Coordinate system management manages the coordinate system of the current robot.

Area ③ is used to set the reference coordinate system when the robot is jogging:

- User: When jogging, the robot TCP moves along the current user coordinate system. For example, X+ indicates that the robot TCP moves along the current user coordinate system. The system moves in the positive X-axis direction.
- Tool: When jogging, the robot TCP moves along the current tool coordinate system. For example, X+ means that the robot TCP moves along the positive X-axis of the current tool coordinate system .

The robot's 3D model will display the currently active coordinate system.

Edit point value

In the motion control area for joint jogging or coordinate system jogging, double-clicking any value will bring up a prompt box **to edit the jogging value .**

Directly input the joint angle or coordinate coefficient value for the target position; the blue outline shading displays the shape of the target position. Long press **to move** the rob ot to the target position.

Target attitude

By using the target posture, the robot can move quickly to a specific location.

Safety origin

Click on **the target posture >** The robot will return to the safe **origin point** when the button is pressed. Press and hold the button to move the robot **to the target position .** Release the button to stop the robot from moving.

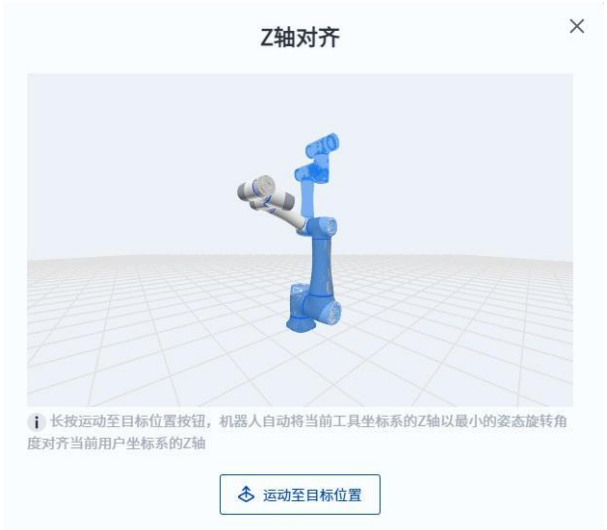
Z- axis alignment

Click on **the target posture > Z-axis alignment** will bring up the following prompt box; pr ess and hold to **move to the target position , and the robot will automatically**

align the Z-axis of the current tool coordinate system with the Z-axis of the current

user coordinate system with the smallest possible rotation angle; release the button

and the robot will stop moving.



Packaging posture

Click on **the target posture > When the robot is in packing posture** , the following prompt box will pop up; long press to move the robot to the packing posture **when the target**

position is reached , and release the button to stop the robot from moving.

Custom

Click on **the target posture > Customizable** ; see the instructions for editing jog values for t he operation interface .

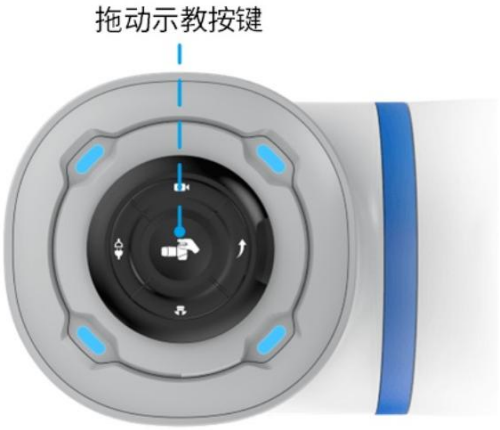
5.8 Drag

The robot can enter drag mode by dragging the teaching button at its end effector. In drag mode, the user can drag the robotic arm to change its posture and perform point teaching.

The resistance level of each joint in drag mode can be modified on the drag settings page.

> **illustrate :**
> the robot , please ensure that the load parameters and installation angle are set correctly.

- The location of the drag-and-teach button for the CRA/CRAF series (excluding CR20A/CR 20AF) robots is shown in the figure below.



The robotic arm is enabled (indicator light is solid green), press and hold the drag teaching button for more than 1.5 seconds. The robot will enter drag mode

(indicator light will flash green quickly). At this time, the user can drag the robot by hand to change its posture.

drag mode, press and drag the teach button briefly (less than 1.5 seconds) to exit drag

mode.

- The drag-and-teach button for the CR20A/CR20AF robot is located on the side of the end effector and is silkscreened with **"FREE"** , as shown in the picture below.

The robotic arm is enabled (indicator light is solid green), press and hold this button to enter drag mode (indicator light flashes green quickly). At this time, the user can drag the robot by hand to change its position.

Release the drag teaching button to exit drag mode.

- The location of the drag-and-drop teaching button on the new version of the Nova robot is shown in the image below.

The robotic arm enabled, press and hold the drag-and-teach button for 3 seconds and then release it to enter drag-and-teach mode. After dragging the robotic arm to the teach point, press the drag-and-teach button again briefly to exit drag-and-teach mode.

- Magician The location of the drag-and-drop teach button on the E6 robot is shown in the figure below.

With the robotic arm enabled (indicator light solid green) , press and hold the drag teaching button for more than 1.5 seconds to enter drag mode (indicator light flashes green rapidly). Users can then drag the robot manually to change its pose. Pressing and holding again in drag mode will enter trajectory recording mode.

Drag mode , press and drag the teaching button briefly (less than 1.5 seconds) to exit drag mode and track recording mode.

5.9 Emergency stop and recovery

In case of emergencies during robot operation, the emergency stop function can be used to stop the robot immediately. Users can trigger the emergency stop function using the following methods:

- Press the emergency stop button connected to the robot controller.
- Trigger user emergency stop input in safe I/O .
- Click the emergency stop button in the upper right corner of the software interface.

After an emergency stop event is triggered, the robot has the following two stopping states:

- If the robot stops within 500ms, it will only be enabled, not powered off.
- If the robot is still moving after 500ms, it will be forcibly de-enabled and powered off.

> ⚠ **Notice :**
> Magician The E6 series robots will power down immediately after an emergency stop event is triggered.

The two states will generate corresponding alarms.

An emergency stop occurs , the emergency stop button icon will flash. To reactivate the robot, first click the emergency stop button again to reset it , then clear the alarm, and finally reactivate the robot (if the robot is powered off, you need to power it on according t o the pop-up prompts).

⚠ Notice :

- The software interface is only a supplement to the hardware emergency stop function. In an emergency, the hardware emergency stop function should be used first to stop the robot.
- Physical emergency stop button or a safety I/O will also change the state of the button, but in this case, the button cannot be reset through the software interface and must be reset through the corresponding input source.

5.10 Speed adjustment

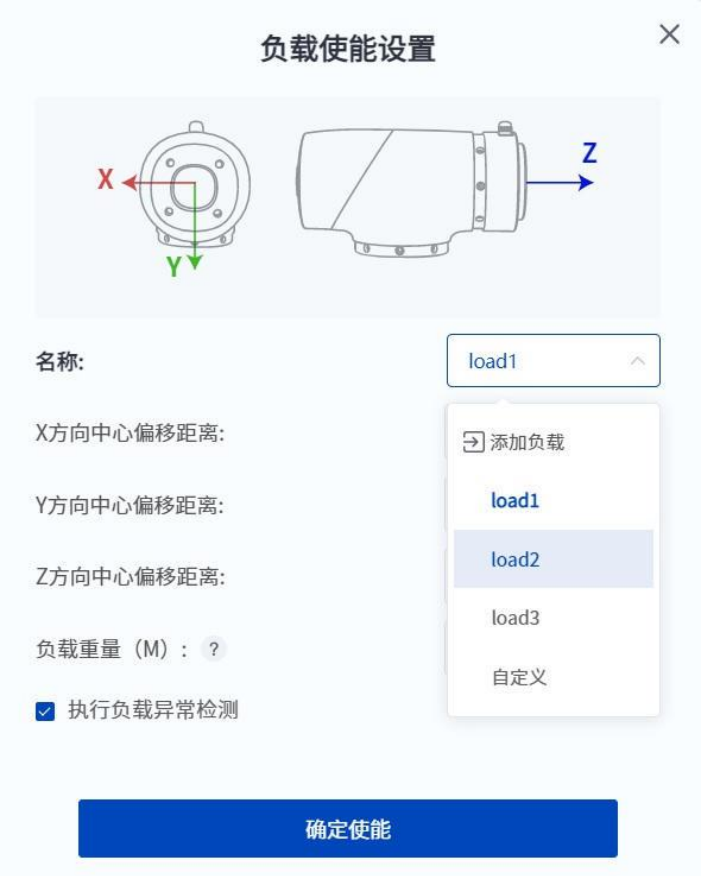
Using the speed slider on the right side of the top toolbar, such as reducing the robot speed during program debugging to ensure operational safety .

5.11 Load settings

The load parameters are the center of mass and weight parameters of the end effector load (including fixture) of the robotic arm. They need to be set according to the actual load. If the settings are incorrect , it will lead to a decrease in the performance of the robotic arm and may cause problems such as false collision detection and uncontrolled dragging.

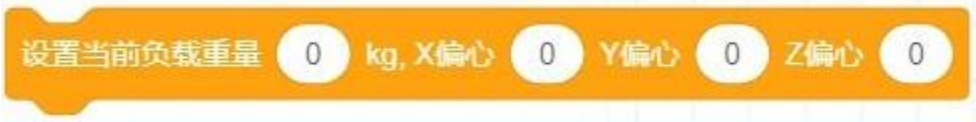
Users can set the load parameters used by the robot using the following methods:

- When enabling the robot via software, you need to set the load parameters in the pop-up window. See the enabling operation instructions for details.



- During programming , corresponding blocks or script instructions can be invoked to s et load parameters during project runtime. See the instruction documentation in the a ppendix for details . Load parameters set using this method only take effect during pr oject runtime.

  Building Blocks :



- **script :**

```
SetPayload(payload, {x, y, z}) -- Custom load parameter
SetPayload(name) -- Use the preset load parameter set
```

Preset load parameter groups are managed on the load parameter page.

5.12 Collision detection settings

The robot will automatically stop upon detecting a collision during movement. Users can configure the collision detection sensitivity and the specific handling method after a collision. For CRA/CRAF/new Nova series robots, please refer to the safety restrictions for collision detection settings ; for Magician... For details on collision detection settings for the E6 series robots , please refer to the Collision Detection section .

Users can also set the collision detection sensitivity during project runtime using corresponding blocks or script commands; collision detection parameters set in this way only take effect during project runtime. For block programming, see the appendix for details on **setting the collision detection function** and **setting the collision backoff distance for** controlling block groups ; for script programming, see the appendix for details on **the SetCollisionLevel** and **SetBackDistance** commands for motion parameters .

5.13 Handling alarms

If the jogging or point-saving method is incorrect or the robot is used improperly, such as by limiting the robotic arm or placing it at a singular point. If an alarm is triggered during the operation of the robotic arm , a red dot with a number will appear in the upper right corner of the log icon, indicating the current number of alarms.

The robot triggers an alarm, the current alarm page will display an alarm level icon, alarm occurrence time, error code, alarm type, description , and solution. Please refer to the description and solution to resolve the alarm issue.

The meanings of the robot alarm level icons are as follows:

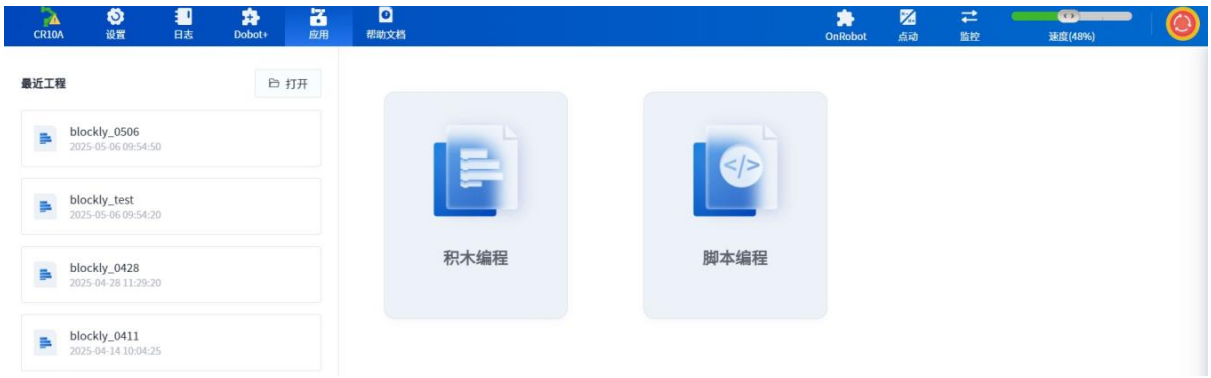
Alarm icon	Alarm Level	Alarm Categories	Corresponding measures
	0	Fault Class	Error message , pause operation, enable and power off.
	1	Exception Class	Error message , pause operation and enable.
	5	Error Class	Error message and paused operation
	1 0	Fault Class	Error message , stopped running, enabled and powered down.
	1 1	Exception Class	Error message , stop running and enable.
	1 5	Error Class	Error and stop running

Click **Clearing alarms** will clear the current alarm. Some alarms cannot be cleared directly. Please refer to the alarm solutions for details.

6 application

6.1 Select application type

Users can choose a suitable method to create a project in **the application interface to control the robot to run automatically.**

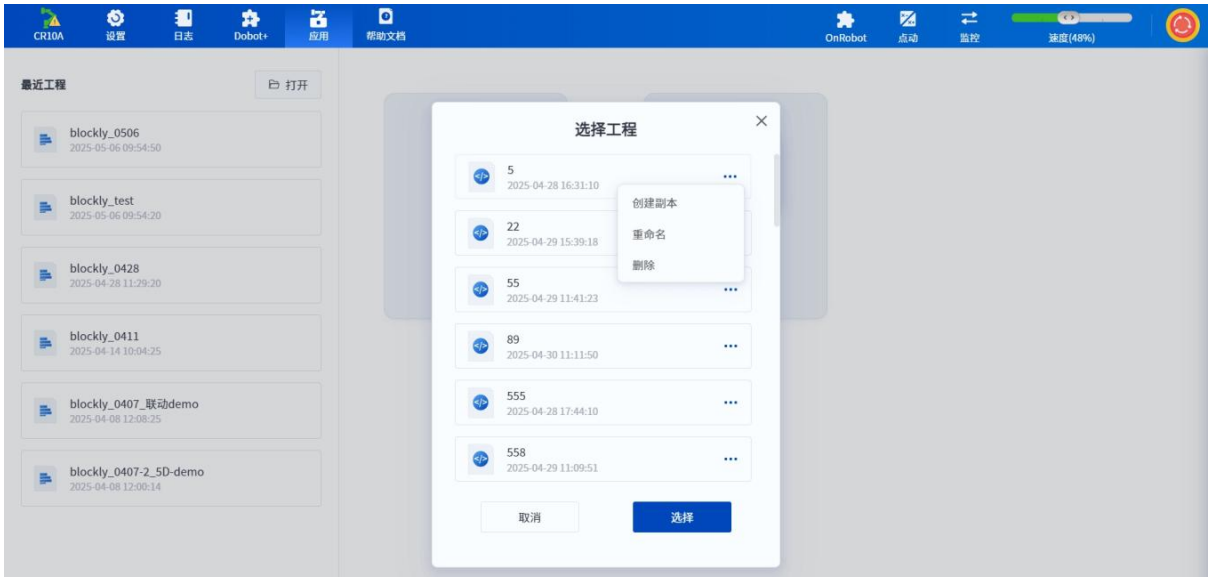


- Block programming : A graphical programming approach that is simple and easy to learn.
- Script programming : A programming method based on the Lua programming language, suitable for users with some programming experience.
- Python Programming : A programming approach based on the Python programming language, **connecting to Magician only on the PC side. It can be used with the E6 robot** for teaching and research purposes.

"Recent Projects" will display the projects you have recently opened. The icons to the left of t he project names have the following meanings.

- Block programming engineering.
- Script programming engineering.
- Python programming project.

Click the **"Open** " button to bring up the project selection dialog box, then click the project name. **...** allows you to create a copy of the current project, rename the current project, or delete the current project.



6.2 Block Programming

6.2.1 Overview

CBSH Studio Pro offers the ability to program blocks. Users can simply drag blocks from the sidebar into the programming area according to their shapes to create programs without writing code.

Projects : Block programming allows editing and running in project units, supporting debugging. Accessing the programming page via the application's homepage icon will automatically create a new, unnamed project. After programming, you must name and

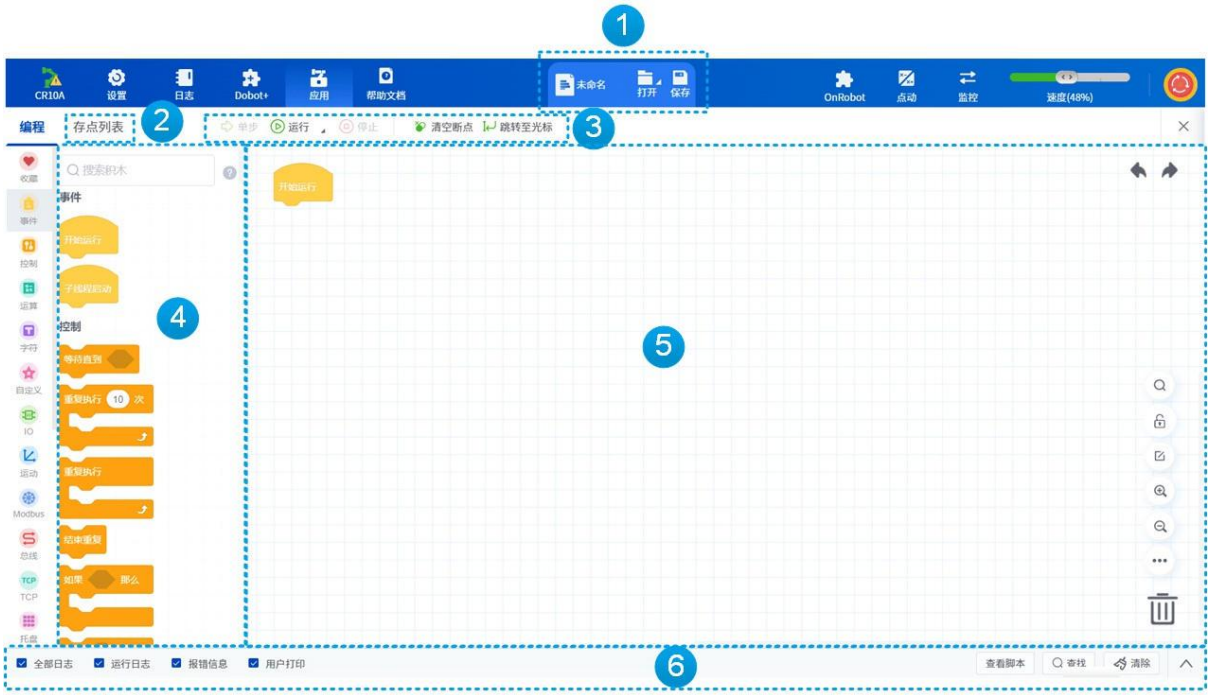
save the project before debugging and running it. Projects are saved in the robotic arm controller, supporting importing and exporting.

Each project supports one main thread and up to four child threads.

- **Main thread** : The main thread uses the **starting** block as its initial block. All blocks can b e used in the main thread.
- **child threads** : child threads use **the child thread startup** block as their starting block. Child threads are parallel threads that run alongside the main thread and can be used to set I/O, variables, etc., but cannot use motion blocks.

Teaching Points : During programming, users can move the robot at any time by jogging or dragging, and then open the teaching point list to save the robot's current pose as a teaching point. Teaching points in the teaching point list are bound to the project and can be used as parameters for commands. If you want to save teaching points that can be called in multiple projects, please use global variables .

the block programming is shown in the figure below.



Serial Number	illustrate
1	Used to display project names and manage projects.
2	Click to open the save point interface, which is used to manage save points in the project.
3	Buttons related to engineering debugging and operation.
4	Provides building blocks needed for programming . You can find the building blocks you need by category and color, or save or search for specific blocks. ? You can view the detailed instructions for the building blocks.
5	Block programming, that is, the area for writing block projects. Modified but unsaved blocks will appear on the left side. An icon indicates to the user that the block has been modified.
6	The runtime log section is used to view the project's runtime logs. Click the rightmost button The icon can expand or collapse the log display are • on the left can be used to filter the • Displayed log types • Click **to view the script** to see the script for the current block program • Click **Find** the specified character in the searchable log • Click **Clear** the area where logs can be displayed

The canvas have the following meanings.

icon	illustrate
	Used to undo/redo programming operations.
	Used to search for all blocks that match the keywords within the current program and subroutines, making it easy to quickly locate blocks.
	Used to lock/unlock the programming area.
	Used to enter edit mode; see the programming instructions below for details.

	These are used to zoom in and out of the programming area, respectively.
	These are used to return to the top of the block, center the block, and ret urn to the bottom of the block, respectively .
	the programming area here to delete it. Users can also right-click the block and select delete.

Block storage function

Right- click on a block in the block box and select **"Add to Favorites"** . The block will be added to the top " **Favorites"** list for easy and quick access .

illustrate :

You can only save blocks from the categories of **Control, Operation, Character, IO, Motion, Modbus, Bus, TCP, and Tray . Blocks from** other categories cannot be saved.

saved blocks are sorted by the time they were saved, and blocks that have already been saved will not be saved again. Saved blocks are cached locally and can still be used the next time you enter the block programming environment.

Right- click on a block in **the Favorites category and select "Remove from Favorites** ." Th e block will be removed from the Favorites category. If there are no blocks in the

Favorites category , the category will be automatically hidden.

6.2.2 Engineering Management

You open the block programming interface, a newly created blank project page is displayed by default, and the project name is displayed as **unnamed** .

Click The program allows users to open a file menu, supporting functions such as creating, opening, saving as, importing, and exporting projects. It also supports converting block projects into script projects. Once converted, block projects can be opened and edited within the script programming interface .

illustrate :

creating a new project, you can choose a blank project or select a block programming template.

In the following scenarios, CBSH Studio Pro will automatically back up your project:

- CBSH Studio Pro checks the currently open project every ten minutes for any modifications; if any are found, it automatically backs up the current project to the controller.
- Before the project starts running, CBSH Studio Pro checks if the currently open project has been modified; if so, it automatically backs up the current project to the controller.
- CBSH Studio When Pro disconnects from the controller (either intentionally or abnormally), CBSH Studio Pro checks if the currently open project has been modified; if so, it automatically backs up the current project to the local device (PC or tablet).
- If saving or saving as a project fails, the current project is automatically backed up to the controller.

Projects backed up to both the controller and local storage can be accessed via... Open the **"Open Backup Project" option** in the menu .

Click You can save the current project. If the project is unnamed, you need to enter a project name first. If saving or saving as fails , a pop-up window will prompt the user with the reason for the failure, and the current project can be found by **opening the backup project** .

6.2.3 Save points

Users can move the robot to the desired pose by tapping or dragging , and then save the position in the save point list.

> **illustrate :**
> Storage point list is open, you can also add teaching points by pressing **the POINT button** on the side of the CR20A/CR20AF end flange .

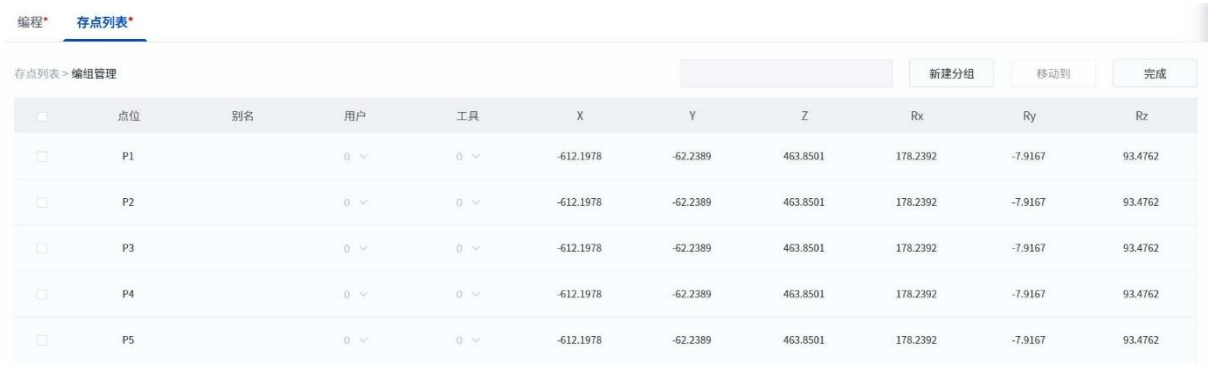
Serial Number	illustrate
1	● Click ┼ **The Add Point** button can save the robot's current pose as a new teaching point. ● Selecting a teaching point, click **The cover** button allows you to cover the point using the robot's current pose. ● Selecting a teaching point, click **he delete** button can be used to delete the teaching point. ● Clicking **"Group Management"** allows you to group the points in the storage list, as described below. ● Clicking **"Alias Filter"** allows you to filter by alias and display points that meet the criteria in the list.
2	Teaching point list. Any value other than **the point location** can be modified by double-clicking or clicking the drop-down list.
3	Control the robot to move to the currently selected point in a specified p attern.

illustrate :

Aliases for locations must be no more than 20 characters long and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters , Russian characters, underscores, and hyphens.

Group Management

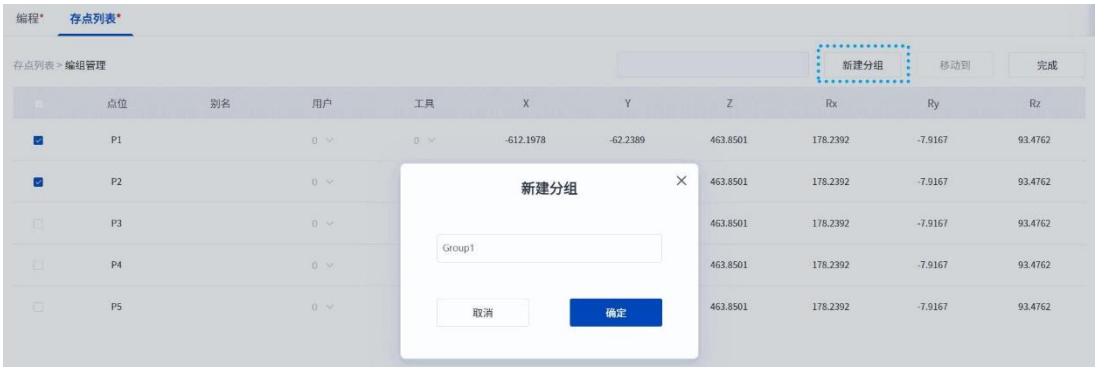
Clicking **the group management** button allows you to create a new group for th e selected points or move them to a specified grou



Create a new group

1. Manually select locations (multiple selections are possible), click the

"Create New Group" button, and a pop-up prompt box for naming the new group will a ppear.



2. Customizing the group name, click the **OK** button, and the selected

points will be assigned to the corresponding group.

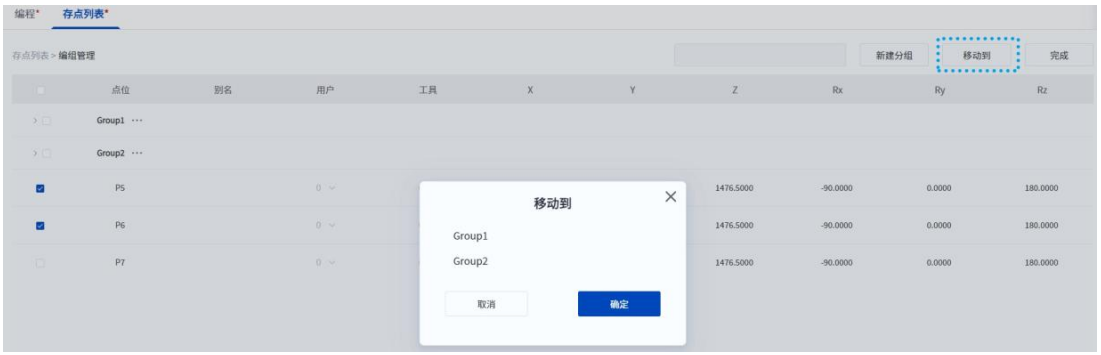
illustrate :

- Clicking the " **...**" after the group name allows you to rename or cancel the c urrent group; after canceling the current group, the points within the group will return to an ungrouped state .
- New group names must be no more than 20 characters long

and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters, Russian characters, underscores, and hyphens. Group names cannot be duplicated.

3. Click the **"Finish"** button to exit group management.

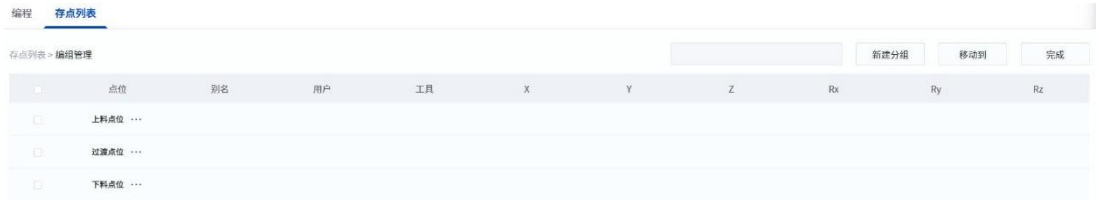
- **Adjusting the sorting of points** : The sorting of points can be adjusted in the following two ways.
  - On **the grouping management** page, manually select points (multiple selections are allowed), and click the **"Move to" button to bring up the gro up selection; you can then** move points to a specified group.



- On **the point list** page, long press and drag the points to adjust their sorting.

- **To add a new point to a specified group** : On **the point list** page, first select the group, then click **"Add Point** ". New locations will be automatically added to the selected group.
- **Grouping Management Example** (Using Loading and Unloading Application as an Example)

  1. Click on **Group Management** , create three new groups, and name them Loading Point, Transition Point, and Unloading Point respectively.



  2. Click " **Finish"** to return to the save point list.
  3. Select the corresponding group and click " **Add Point"** to quickly add the new point t o the corresponding group.

View location

On **the point list** page, the pose of the selected point will be displayed in blue outline in the simulation area of **the jog** panel, as shown in the figure below.

6.2.4 programming

Block Programming Demo

You begin programming , please clearly define the functionality your program aims to achieve. This section uses the example of writing a program to control a robot to move cyclically between two points to introduce how to write a modular program.

> ⚠ **Notice :**
> starting debugging or running the project, ensure that there are no people or any other obstacles within the robot's working range.

1. From the left-hand block area **, control** the block group and drag **the repeat block below the starting block** on the canvas .
2. Drag **the Motion to Point** block from the **Motion Blocks group to the Repeat Execution** block, click the Point drop-down box, and select P1.
3. Drag another **Motion to Point** block below the previous **Motion to Point** block, click the Point drop-down box, and select P2 .

At this point , a simple loop motion program has been written.

Right-click /long-press the block to support the following functions

In addition to the basic programming operations mentioned above , right-clicking a block on the canvas (long press on mobile devices, the same applies below) to open the menu also supports the following programming operations:

- **Copy and Paste** : This copies the selected block and all blocks connected to it below it (if the selected block contains nested blocks, those will also be copied). The copied block will appear under the mouse cursor. Move the block to the target location. Click the left mouse button to Paste the copied block into the target location.
- **Copy to clipboard** : Copies the selected block and all blocks connected to it below it (If the selected block contains nested blocks, those will also be copied). The copied

blocks are cached to the clipboard. Right-click at the target location and select **paste** . The copied block will hover below the mouse cursor; left-click to paste the copied block into the target location.

- **Copy a single block and paste** : This function copies the selected single block (if the selected block contains nested blocks, those will also be copied). The copied block will hover below the mouse cursor. Move the block to the target location. Click the left mouse button to copy. The blocks are pasted into the target location.

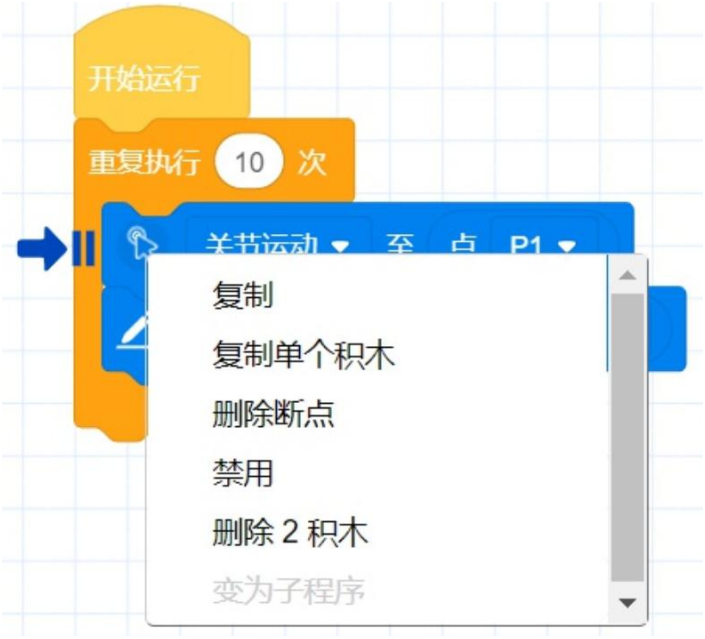
- **Copy a single block to the clipboard** : Copy the selected single block (if the selected block contains other blocks, they will all be copied) . The copied block is cached to the clipboard. Right-click at the target location and select **Paste .** The copied block will be hovering below the mouse cursor; left-click to paste the copied block into thet arget location.

- **Add /Delete Comments** : Right-click the block and select **"Add Comment** " to display the floating comment box. On the right side of the block , you can drag its position or size. Click to enter your comment, then click ▼ to collapse the comment; after collapsing, click ▶ to expand the comment again .

  Click the upper right corner of the comment box **X** or right-click and select **"Delete Comment"** .

- **Adding /Removing Breakpoints** : For blocks that support breakpoints, right-click and select " **Add Breakpoint** ." The breakpoint marker  is shown in the image below . When the program reaches a block with a breakpoint, execution will pause; the cursor will remain on the breakpoint block, but the breakpoint block will not be executed.

  For a block that has had breakpoints added, right-clicking and selecting **"Delete Breakpoint"** will remove the breakpoints from the current block.



illustrate :

- While the program is paused, breakpoints can be added or removed.
- Only when block programming **starts running** , and the blocks under **the subroutine support adding breakpoints.**

If you add breakpoints to the entire subroutine (but not to the blocks within the subroutine), then the subroutine blocks will display :

- If breakpoints are also added to other blocks within the subroutine, then the subroutine blocks will display:

- If a block with added breakpoints is dragged back to the menu bar, dragged out of **the Start menu** , or outside of **a subroutine** , the breakpoint function will automatically be disabled .

- **Disable** : Select the block, right-click and choose **Disable** ; the current block is disabled and turns gray. You can view the script of the disabled block by **clicking the Script** button, which will then be marked as a comment.

  Disabled blocks cannot have annotations or breakpoints added to them . Blocks with breakpoints already added to them will have their breakpoints cleared when they are disabled.

- **Deletion** : Users can delete blocks in the following five ways.

  1. Drag the blocks to the left block box and then release.
  2. Drag the block to the trash can icon in the bottom right corner of the canvas and release.
  3. Right-click on a block and select **Delete** (if the selected block contains other blocks, they will all be deleted together).
  4. Click on the  right side of the canvas.

     The icon enters edit mode. Manually select blocks (multiple selections are possible) , and click the delete button above the canvas to delete the selected blocks.
  5. Right-clicking on an empty space in the canvas and selecting delete will delete all blocks in the current canvas.

- **To convert a block to a subroutine** : Right-click on a block that is not connected to **Start** or **Subthread Startup** to open the menu, and select **Convert to Subroutine. The `subroutine` option** converts the set of blocks into a subroutine. Once converted, a single subroutine block can replace the entire set , facilitating reuse and improving programming efficiency. **This operation is not available in the subroutine editing interface.**

- **Fold /Expand Blocks** : Right-clicking on nested blocks allows you to fold or expand t hem for easier viewing and editing . Folding blocks clears breakpoints, preventing the addition of breakpoints to folded blocks.

Right-click on a blank area of the canvas (long press on mobile devices, the same below) to open the menu, where you can perform operations such as undo, paste, redo, organiz e, collapse all, expand all, or delete all blocks in the current canvas.

Canvas editing function

Click on the right side of the canvas.
Cancel or delete operations.

Clicking **"Exit Editing"** or performing other operations in the programming area will exit the editing state.

Further information about building blocks is not provided here; please click the upper right corner of the building blocks menu. See Appendix B for details .

6.3 Scripting

6.3.1 Overview

CBSH robots offer a wide range of CBSH products. API interfaces, such as motion commands and TCP/UDP commands, are implemented using the Lua language. This allows users to easily call it during secondary development. CBSH Studio Pro provides a Lua scripting environment, allowing users to write their own Lua scripts to control the robot.

Projects : Script programming is edited and run on a project-by-project basis, supporting debugging. Accessing the programming page via the application's homepage icon will au tomatically create a new , unnamed project. After programming, you must name and

save the project before debugging and running it. Projects are saved in the robotic arm

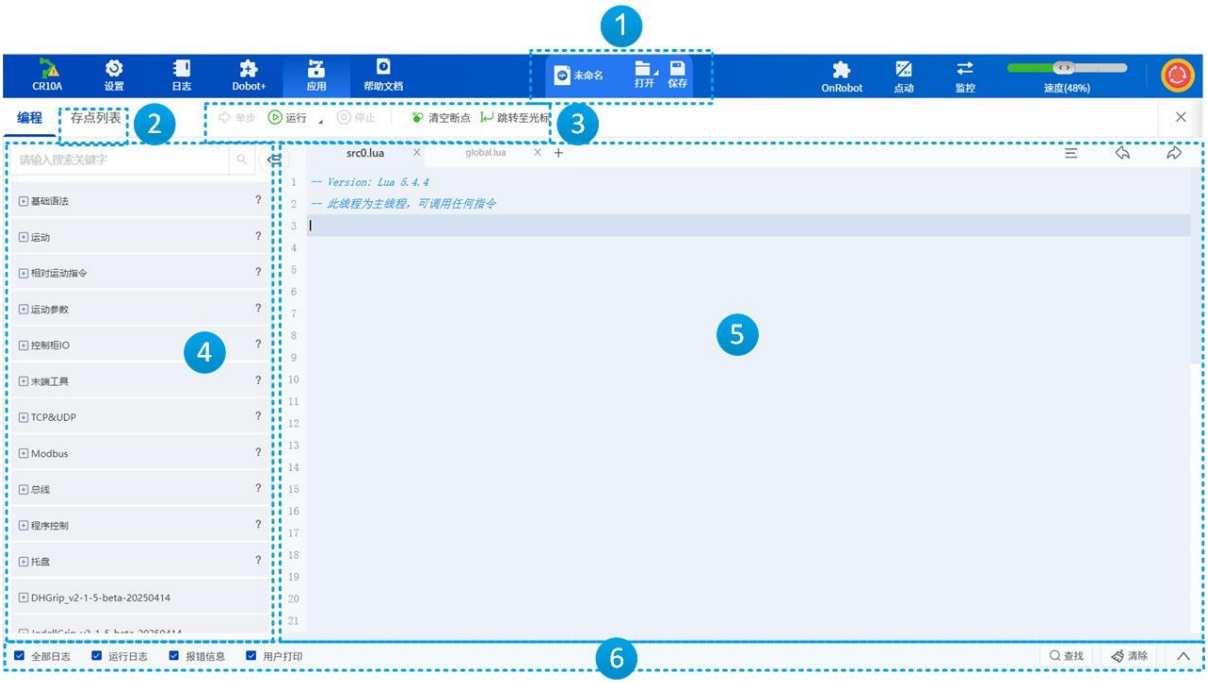
controller and support importing and exporting.

A project will include the following script files, displayed as tabs:

- **The src0.lua file** is the main thread and can call any instruction.
- T h e g lobal.lua file is only used to define variables and sub-functions.
- There are 0 to 4 child **threads** , named **src1.lua** to **src4.lua** . Child threads are parallel programs that run alongside the main program, with a maximum of 4. They can configure I/O, variables, etc., but cannot call motion instructions.

The project starts , the robot will execute the instructions in the main thread and child threads sequentially from top to bottom; the user does not need to define an entry point (main) function . For more basic Lua syntax, please refer to Appendix C.

Teaching Points : During programming, users can move the robot at any time by jogging or dragging, and then open the teaching point list to save the robot's current pose as a teaching point. Teaching points in the teaching point list are bound to the project and can be used as parameters for commands. If you want to save teaching points that can be called in multiple projects, please use global variables .
For script programming is shown in the figure below.



Serial Number	illustrate
1	Used to display project names and manage projects.
2	Click to open the save point interface, which is used to manage save point s in the project.
3	Buttons related to engineering debugging and operation.
4	Used to find and use programming functions. Click ? You can view the function documentation.
5	In Code editing area, you can click once to change script document or click once to add Child thread (On PC, you can also use the shortcut **Shift + Tab**) triggers a script formatting, Script alignment for easier customer viewing. Click the upper right corner.
6	The runtime log section is used to view the project's runtime logs. ● Click the far right ^ can open or collapse the log display area ● On the left can be used to filter the displayed log types; ● Click **Find** the specified character in the searchable log; ● Click **Clear** the area where logs can be displayed.

6.3.2 Engineering Management

The script programming interface is opened , a newly created blank project page is displayed by default, and the project name is displayed as **unnamed** .

Click can be opened , supporting the creation, opening, saving as, importing, and exporting of projects.

illustrate :

Creating a new project, you can choose a blank project or select a script programming template.

In the following scenarios, CBSH Studio Pro will automatically back up your project:

- CBSH Studio Pro checks the currently open project every ten minutes for any modifications; if any are found, it automatically backs up the current project to the controller.
- Before the project starts running, CBSH Studio Pro checks if the currently open project has been modified; if so, it automatically backs up the current project to the controller.
- CBSH Studio When Pro disconnects from the controller (either intentionally or abnormally), CBSH Studio Pro checks if the currently open project has been modified; if so, it automatically backs up the current project to the local device (PC or tablet).
- If saving or saving as a project fails, the current project is automatically backed up to the controller.

Projects backed up to both the controller and local storage can be accessed via...  Open the **"Open Backup Project" option** in the menu .

Click  You can save the current project. If the project is unnamed, you need to enter a project name first. If saving or saving as fails , a pop-up window will prompt the user with the reason for the failure, and the current project can be found by **opening the backup project** .

6.3.3 Save points

Users can move the robot to the desired pose by tapping or dragging , and then save the position in the save point list.

> **illustrate :**
> storage point list is open, you can also add teaching points by pressing **the POINT button** on the side of the CR20A/CR20AF end flange .

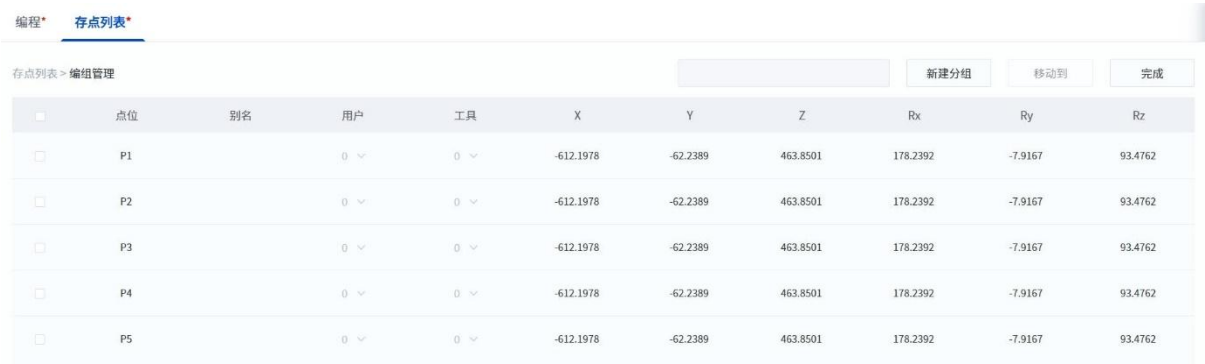
Serial Number	illustrate
1	• Click ┼ **The Add Point** button can save the robot's current pose as a new teaching point. • selecting a teaching point, click **The cover** button allows you to cover the point using the robot's current pose. • selecting a teaching point, click **The delete** button can be used to delete the teaching point. • Clicking **"Group Management"** allows you to group the points in the storage list, as described below. • Clicking **"Alias Filter"** allows you to filter by alias and display points that meet the criteria in the list.
2	Teaching point list. Any value other than **the point location** can be modified by double-clicking or clicking the drop-down list.
3	Control the robot to move to the currently selected point in a specified pattern.

illustrate :

Aliases for locations must be no more than 20 characters long and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters , Russian characters, underscores, and hyphens.

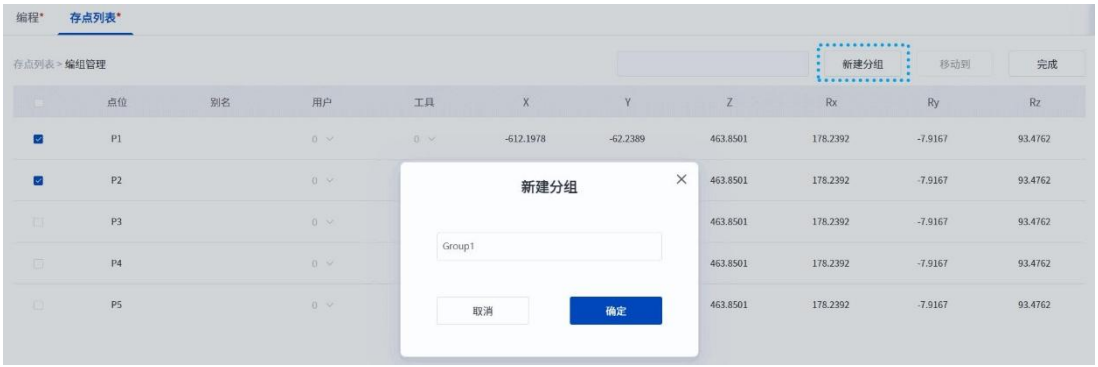
Group Management

Clicking **the group management** button allows you to create a new group for the selected points or move them to a specified group.



- Create a new group

1. Manually select the location, click the **"Create Group"** button, and a pop-up prompt box will appear asking you to name the new group.



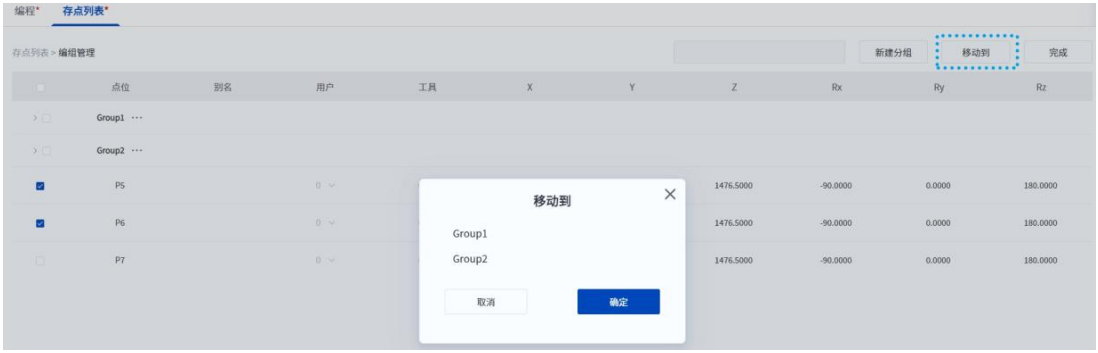
2. Customizing the group name, click the **OK** button, and the selected points will be assigned to the corresponding group.

illustrate :

- Clicking the " **...**" after the group name allows you to rename or cancel the current group; after canceling the current group, the points within the group will return to an ungrouped state .
- New group names must be no more than 20 characters long and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters, Russian characters, underscores, and hyphens. Group names cannot be duplicated.

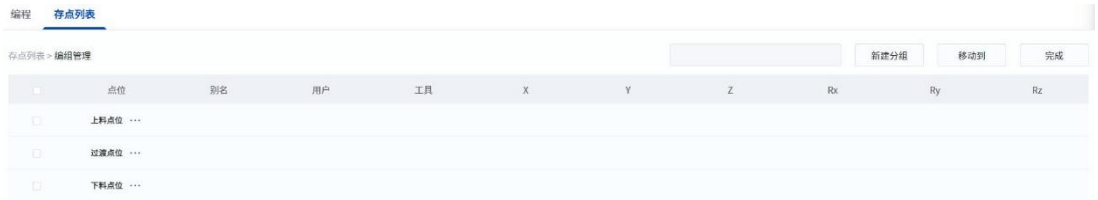
3. Click the **"Finish"** button to exit group management.

- **Adjusting the sorting of points** : The sorting of points can be adjusted in the following two ways.
  - On **the grouping management** page, manually select points (multiple selections a re allowed), and click the **"Move to" button to bring up the group selection; you c an then** move points to a specified group.



  - On **the point list** page, long press and drag the points to adjust their sorting.
- **To add a new point to a specified group** : On **the point list** page, first select the group, then click **"Add Point** ". New locations will be automatically added to the selected group.
- **Grouping Management Example** (Using Loading and Unloading Application as an Example)

1. Click on **Group Management** , create three new groups, and name them Loading Point, Transition Point, and Unloading Point respectively.



2. Click " **Finish"** to return to the save point list.
3. Select the corresponding group and click " **Add Point"** to quickly add the new point t o the corresponding group.

View location

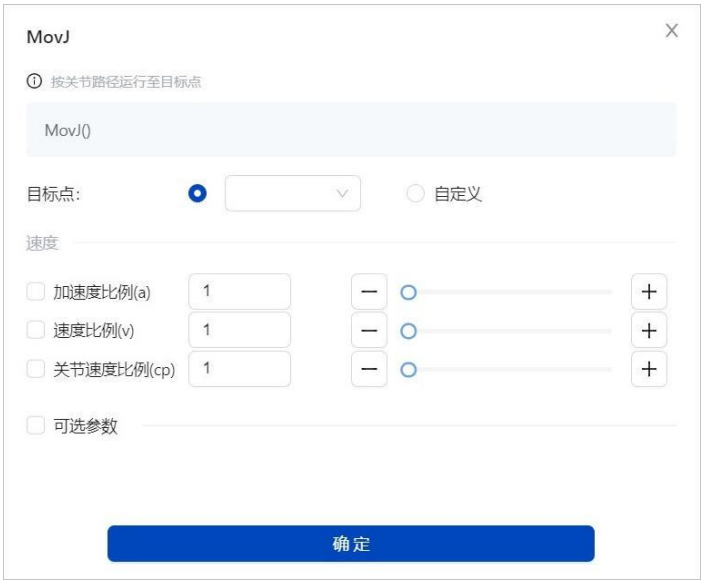
On **the point list** page, the pose of the selected point will be displayed in blue outline in the simulation area of **the jog** panel, as shown in the figure below.

6.3.4 programming

Invoke instructions

Users can invoke commands in the following three ways:

1. Find the function you want to use in the function menu on the left and click on it to t he right. A parameter setting window will pop up. After setting the parameters in t he window and clicking **OK,** a function call instruction with parameters will be added to the cursor position in the programming area.



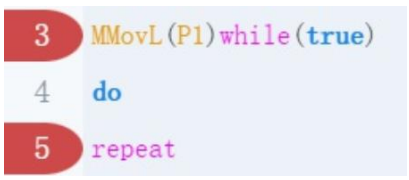
2. Find the function you want to use in the function menu on the left and double-click it. Insert the instruction into the programming area with default parameters, and then modify the parameter values according to your needs .

illustrate :
Enter **The /** key will display the shortcut key **for adding points** and the currently Saved points, allowing you to quickly add points or select existing points.

3. Directly in the programming area on the right. Keyword matching is supported; all commands containing the keyword will be displayed . Keyword matching is case-insensitive.

Other scripting functions

- **Adding breakpoints** : You can add breakpoints by clicking on lines of code with your mouse or finger. Breakpoint markers are shown in lines 3 and 5 of the image below. When the script reaches a line of code with a breakpoint set, execution will pause; the cursor will remain on the breakpoint line, but the breakpoint line will not be executed .



- **Deleting a breakpoint** : Clicking the line of code with an existing breakpoint again will delete the breakpoint, or you can click on all breakpoints in the project . The **"Clear**

Breakpoints " button cannot be used during program execution .

illustrate :

- The program is paused, breakpoints can be added or removed.
- Only the scripting language src0.lua supports adding breakpoints.

Scripting Demo

You begin programming , please clearly define the functionality your program aims to achieve. This section uses the example of writing a program to control a robot to move cyclically between two points to introduce how to write a script.

1. On the save point page, add the starting point P1 and the ending point P2 of the robot's cyclical motion in sequence.
2. `a while` loop function using any of the above **calling instructions .**
3. Add a `MovJ movement instruction` before `the end` of the loop code , with the target point being P1.
4. Add another `MovJ` movement command, targeting point P2. The final code is as follows.

```
while ( true )
do
    Mo vJ(P1)
    Mo vJ(P2)
end
```

At this point, you have successfully written a simple looping program.
If you need to create a subthread, click the "+ Add Subthread" button to the right of the tabs above the programming area, then switch to the Subthread tab to write your program.

Further instructions on script programming are not provided here; please refer to the function menu. See Appendix C for details. .

6.4 Python programming (Magician) E6)

6.4.1 Overview

Magician only connects to PC The E6 robot supports Python programming.

CBSH robots offer a wide range of CBSH products. API interfaces, such as motion commands and TCP/UDP commands, are written in Python, making them easy for users to call during secondary development. CBSH Studio Pro provides a Python programming environment, allowing users to write their own Python programs to control the robot.

Projects : Python programming is edited and run in project form, supporting debugging. Accessing the programming page via the application's homepage icon will default to Create a new, unnamed project. After programming, you need to name and save it before you can debug and run it. The project is saved in the robotic arm controller and supports importing and exporting.

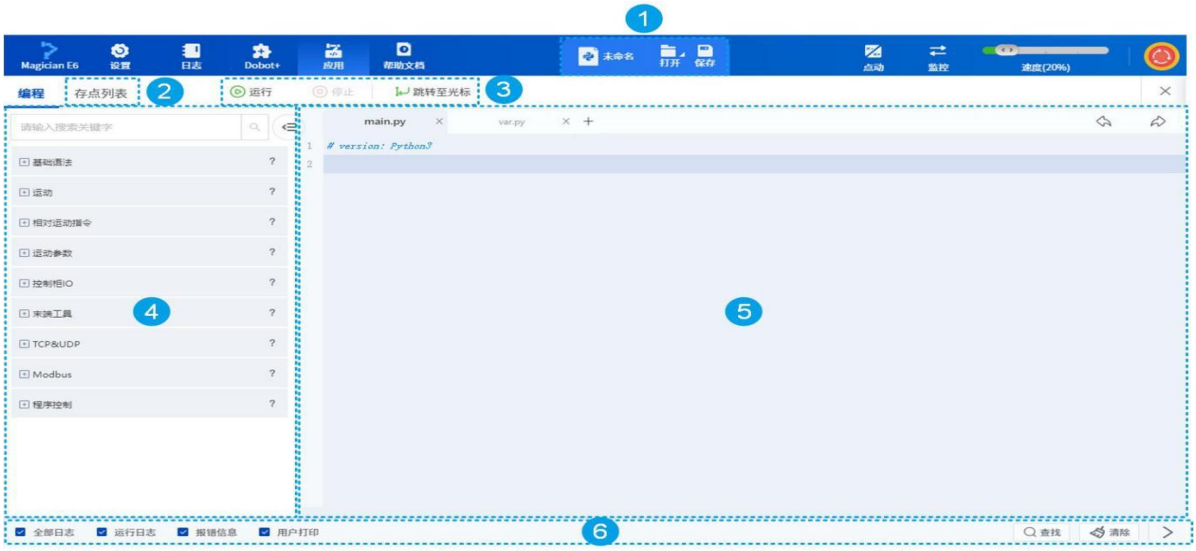
A project will include the following Python files, displayed in tabs:

- **The main.py file** is the main thread and can call any instruction.
- **The var.py file** is only used to define variables.
- There are 0 to 4 child **threads** , named **script1.py** to **script4.py** . Child threads are parallel programs that run alongside the main program, with a maximum of 4. They can have I/O and variables configured, but cannot call motion instructions.

The project starts, the robot will execute the instructions in the main thread and sub-threads sequentially from top to bottom; the user does not need to define an entry point (main) function. For more basic Python syntax, please refer to Appendix E. .

Teaching Points : During programming, users can move the robot at any time by jogging or dragging, and then open the teaching point list to save the robot's current pose as a teaching point. Teaching points in the teaching point list are bound to the project and can be used as parameters for commands. If you want to save teaching points that can be called in multiple projects, please use global variables .

Python programming is shown in the figure below.



Serial Number	illustrate
1	Used to display project names and manage projects.
2	Click to open the save point interface, which is used to manage save points in the project.
3	Buttons related to project operation.
4	Used to find and use programming functions. Click ? You can view the function documentation.
5	In the code editing area, you can click the tabs to switch between Python files, or click + to add a subthread. Click the button in the upper-right corner Reverse/undo programming operations.
6	The runtime log section is used to view the project's runtime logs. ● Click the icon on the far right to expand or collapse the log display area; ● The options on the left can be used to filter the displayed log types. ● Click Find the specified character in the searchable log; ● Click Clear the area where logs can be displayed.

6.4.2 Engineering Management

The script programming interface is opened , a newly created blank project page is displayed by default, and the project name is displayed as **unnamed** .

Click can be opened , supporting the creation, opening, saving as, importing, and exporting of projects.

In the following scenarios, CBSH Studio Pro will automatically back up your project:

- CBSH Studio Pro checks the currently open project every ten minutes for any modifications; if any are found, it automatically backs up the current project to the controller.
- Before the project starts running, CBSH Studio Pro checks if the currently open project has been modified; if so, it automatically backs up the current project to the controller.
- CBSH Studio When Pro disconnects from the controller (either intentionally or abnormally), CBSH Studio Pro checks if the currently open project has been modified; If so, it automatically backs up the current project to the local device (PC or tablet).
- If saving or saving as a project fails, the current project is automatically backed up to the controller.

Projects backed up to both the controller and local storage can be accessed via Open th e **"Open Backup Project" option** in the menu .

Click You can save the current project. If the project is unnamed, you need to ente r a project name first. If saving or saving as fails , a pop-up window will prompt the us er with the reason for the failure, and the current project can be found by **opening the backup project** .

6.4.3 Save points

Users can move the robot to the desired pose by tapping or dragging , and then save the position in the save point list.

Serial Number	illustrate
1	● Click **The Add Point** button can save the robot's current pose as a new teaching point. ● Selecting a teaching point, click **The cover** button allows you to cover the point using the robot's current pose. ● Selecting a teaching point, click **The delete** button can be used t o delete the teaching point. ● Clicking **"Group Management"** allows you to group the points in the storage list, as described below. ● Clicking **"Alias Filter"** allows you to filter by alias and display points that meet the criteria in the list.
2	Teaching point list. Any value other than **the point location** can be modified by double-clicking or clicking the drop-down list.

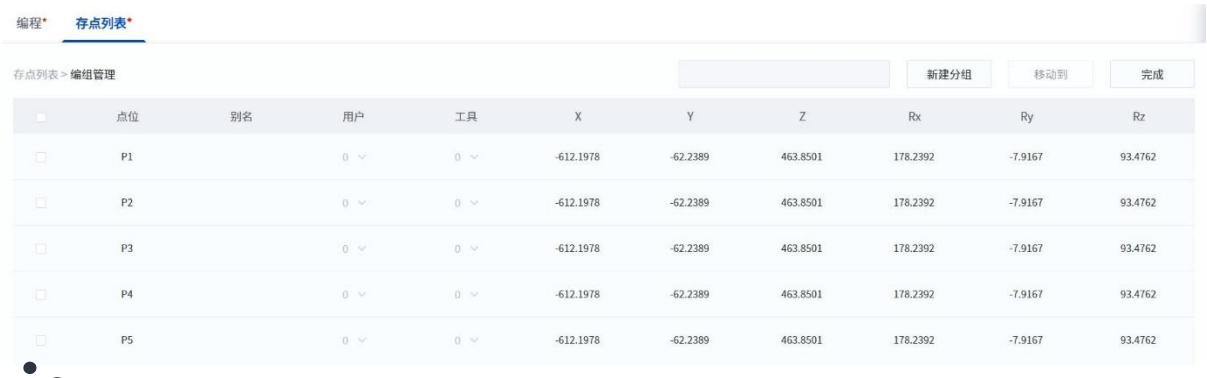
3	Control the robot to move to the currently selected point in a specified pattern.

illustrate :

Aliases for locations must be no more than 20 characters long and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters , Russian characters, underscores, and hyphens.

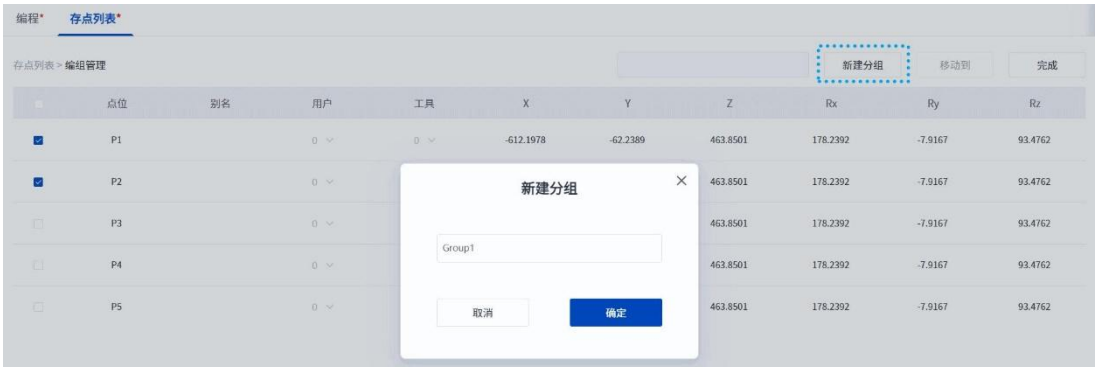
Group Management

Clicking **the group management** button allows you to create a new group for the selected points or move them to a specified group.



- **Create a new group**

1. Manually select locations (multiple selections are possible), click the **"Create New Group"** button, and a pop-up prompt box for naming the new group will appear.



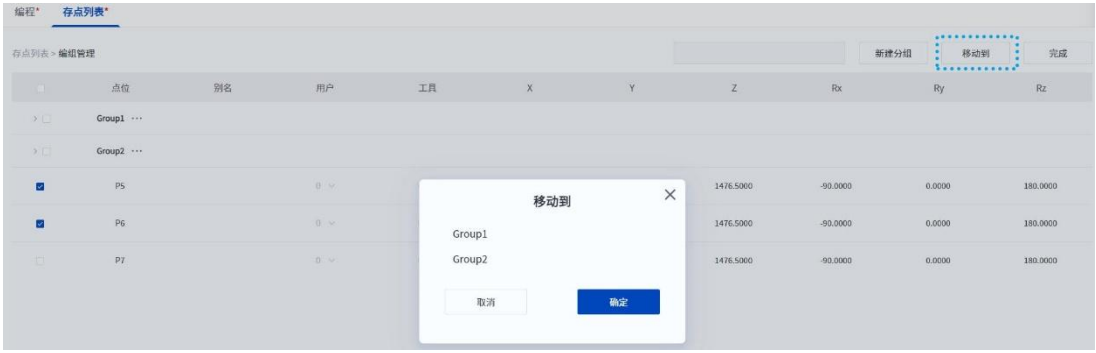
2. Customizing the group name, click the **OK** button, and the selected points will be assigned to the corresponding group.

illustrate :

- Clicking the " **...**" after the group name allows you to rename or cancel the current group; after canceling the current group, the points within the group will return to an ungrouped state .
- New group names must be no more than 20 characters long and can contain letters, numbers, Chinese characters, Japanese characters, German characters, Korean characters, Spanish characters, Russian characters, underscores, and hyphens. Group names cannot be duplicated.

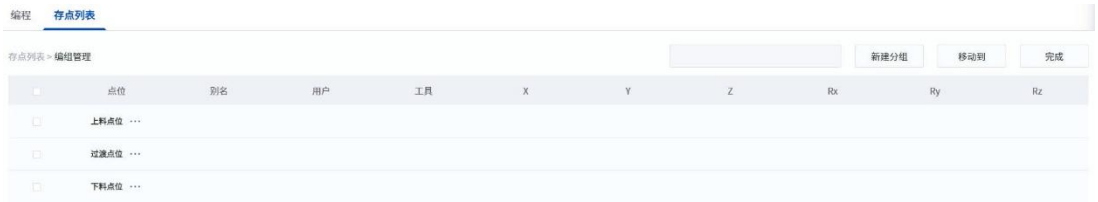
3. Click the **"Finish"** button to exit group management.

- **Adjusting the sorting of points** : The sorting of points can be adjusted in the following two ways.
  - On **the grouping management** page, manually select points (multiple selections a re allowed), and click the **"Move to" button to bring up the group selection; You c an then** move points to a specified group.



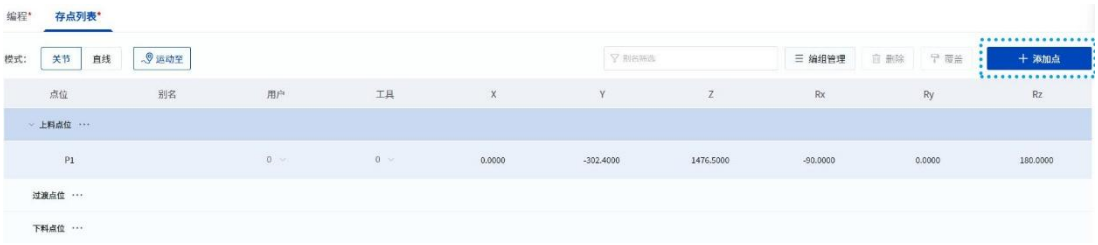
  - On **the point list** page, long press and drag the points to adjust their sorting.
- **To add a new point to a specified group** : On **the point list** page, first select the group, then click **"Add Point** ". New locations will be automatically added to the selected group.
- **Grouping Management Example** (Using Loading and Unloading Application as an Example)

1. Click on **Group Management** , create three new groups, and name them Loading Point, Transition Point, and Unloading Point respectively.



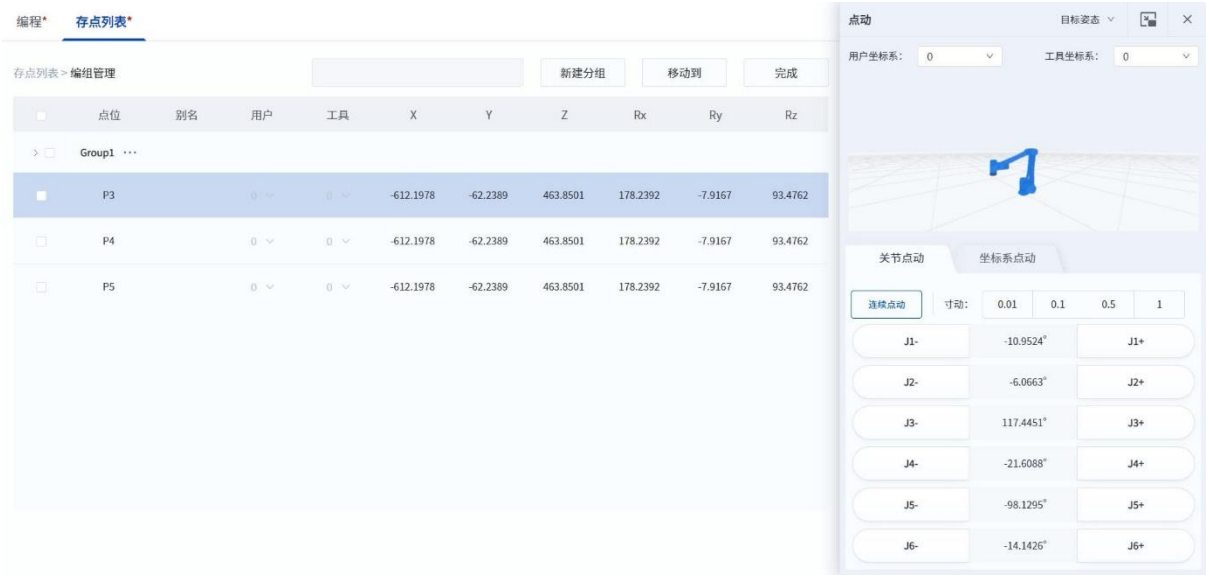
2. Click " **Finish"** to return to the save point list.

3. Select the corresponding group and click " **Add Point"** to quickly add the new point t o the corresponding group.



View location

On **the point list** page, the pose of the selected point will be displayed in blue outline in the simulation area of **the jog** panel, as shown in the figure below.

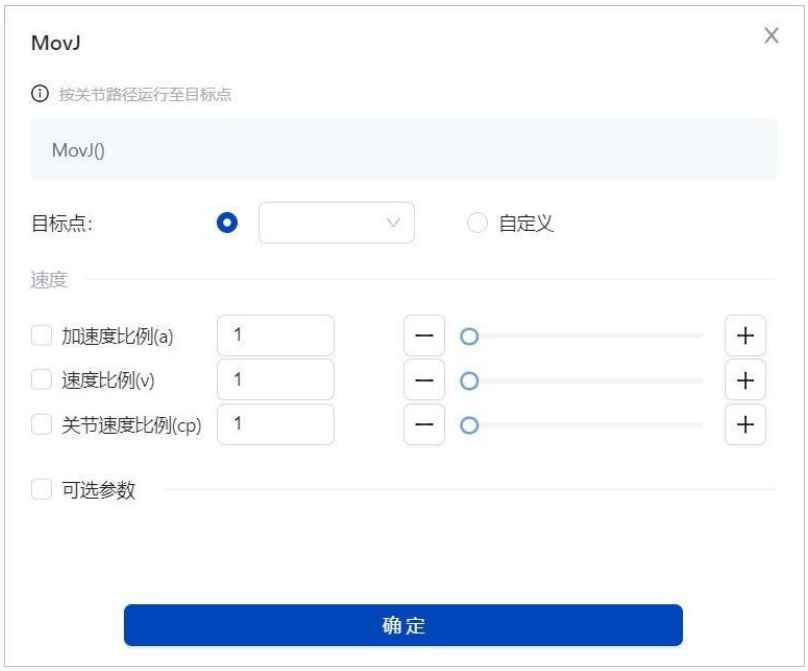


6.4.4 programming

Invoke instructions

Users can invoke commands in the following three ways:

1. Find the function you want to use in the function menu on the left and click on it to t he right. A parameter setting window will pop up. After setting the parameters int he window and clicking **OK,** a function call instruction with parameters will be added to the cursor position in the programming area.



2. Find the function you want to use in the function menu on the left and double-click it. Insert the instruction into the programming area with default parameters, and then modify the parameter values according to your needs .
3. Directly in the programming area on the right, and the program supports automatic command completion via the TAB key on the keyboard.

Scripting Demo

You begin programming , please clearly define the functionality your program aims to achieve. This section uses the example of writing a program to control a robot to move cyclically between two points to introduce how to write a script.

1. On the save point page, add the starting point P1 and the ending point P2 of the robot's cyclical motion in sequence.

2. a while loop function using any of the above **calling instructions .**
3. Add a MovJ movement instruction to the loop code , with the target point being P1.
4. Add another MovJ movement command, targeting point P2. The final code is as follows.

```
while True : Mo
    vJ(P1) Mo
    vJ(P2)
```

At this point , a simple loop motion program has been written.

If you need to write a sub-thread, you can click the tab on the right side above the programming area. **+** Add a child thread, then switch to the child thread tab to write the program.

Further instructions on script programming are not provided here; please click on the function menu ? to view them. The content is the same as in Appendix E. .

6.5 Debugging and running (block/script programming)

> ⚠ **Notice :**
> Starting debugging or running the project, ensure that there are no people or any other obstacles within the robot's working range.

Single-stepping, running from selected lines, single-stepping from selected lines, and clearing breakpoints are only applicable to block-based programming and script programming. The following uses block-based programming as an example to illustrate the debugging and running functions of each icon:

icon	illustrate
single step	While paused click the **The single- step** button will finish running the currently highlighted line, and the cursor will pauses and single-step commands end when switching to the next line or another line (in cases of function entry, return, goto, etc.) .
run	Open the project, click **"Run** " button to start the program with one click. At this time, you must disable to use **the single- step** and **clear breakpoint** buttons to prevent adding or deleting breakpoints.
Run from selected lines	● The program will start running from the selected row. ● Selected line cannot be navigated to (such as a blank line, comment line, or other line without instructions),the software will display a pop-up message indicating failure and will not start the program.
Step through the selected rows	From the selected line, and enter a paused state after execution.
Stop	Click to stop the project from running.
Clear Breakpoints	●Clear all breakpoints in the current project (for instructions on how to set breakpoints, see Block Programming and Script Programming). ●Do not use this while the program is running. ●Creating a new project, opening a project, or saving as will clear the workspace breakpoints.
Jump to cursor	Debugging or operation, the interface will highlight the currently executing instruction (as shown in the block programming example below). Users can click... **Jump to the cursor** button to quickly locate the highlighted line. 

illustrate :

- For script programming, only src0.lua supports **running from a selected line** and **stepping through a selected line** ; the selected line refers to the line where the inPut
- cursor is located, corresponding to the program line highlighted in blue.
  For block programming, only blocks under the **"Start Running " option supportrunning from the selected row** and **stepping through the selected row** ; the selected row refers to the selected block being thickened and widened (long press for 300ms on mobile devices, long press for 100ms on PC).

Program monitoring

Users can monitor the running project through program variables or **by jumping to the cursor . External signal execution (block programming/script programming)**

Users can also run specified projects via external signals (IO or Modbus). If the user is editing a project, the software will display a pop-up window. The system prompts the user to perform a backup. The user can choose to back up or not, and will be returned to the software homepage. If the project the user is currently editing is the same project that needs to be run , the robot will run the last saved project. To run the content being edited, please open the backup file after stopping the project triggered by external signals , complete the editing, save it, and then run the program.

An external signal triggers a project to run, the software will pop up a window indicating that a project is in progress. Users can choose to stop the project or enter the running project to view and control its status.

6.6 Trajectory recovery

The robot is paused, it can automatically return to the pause point using the trajectory recovery function and continue to operate normally according to the original program.

The track recovery function is enabled by default and cannot be disabled.

If **the pause mode supports the jog** button, after the robot triggers the pause event, operations such as jogging, dragging, switching manual mode, switching automatic mode, enabling up, and enabling down are allowed on the robot.

> ⚠ **Notice :**
> The robot enters a pause state after triggering an alarm (including alarms caused by emergency stops).

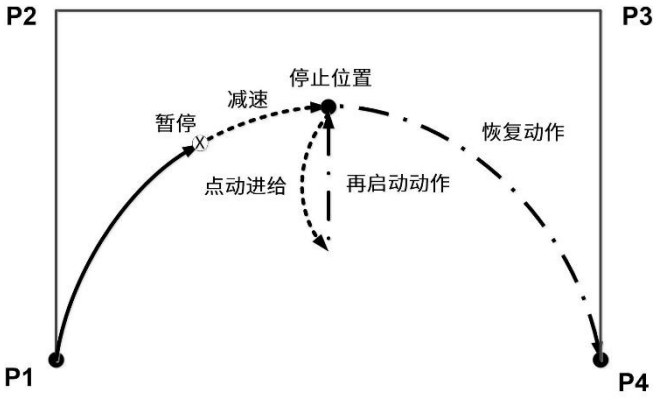
illustrate :

You can determine whether the application is currently paused or stoppedby looking at the style of the **application icon on the main interface:**

- It indicates that it is currently paused.
- This indicates that the current state is stopped.

Enable trajectory recovery effect

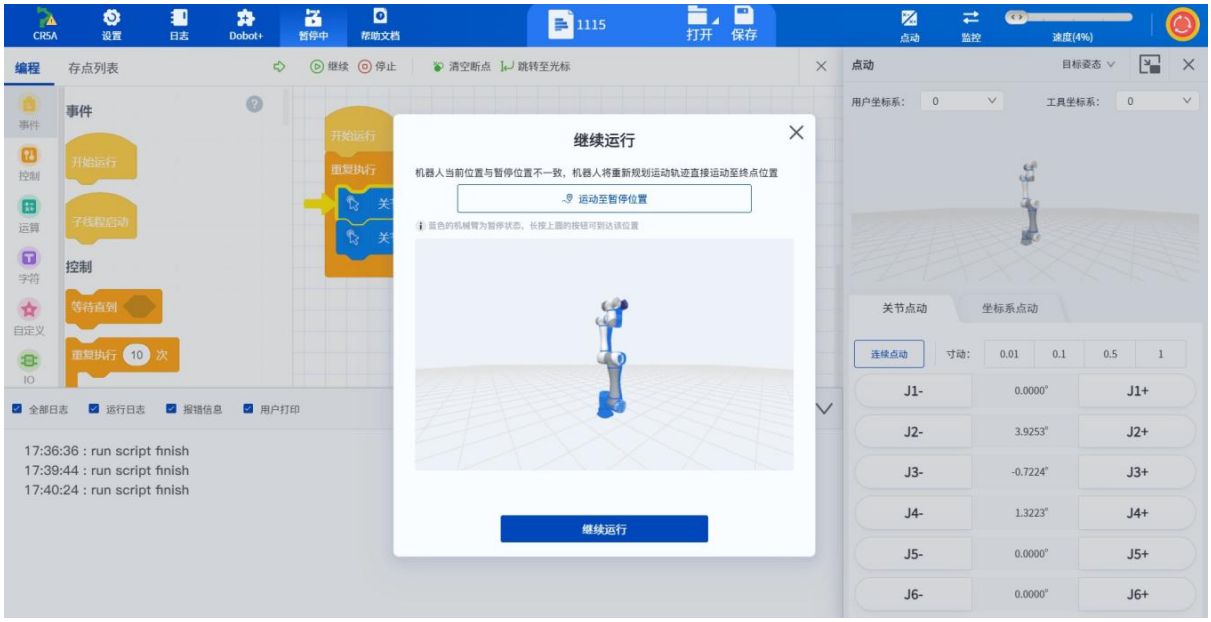
When the robot is paused, activating the trajectory recovery function causes the robot to slowly move back to the paused position at a jog speed, then resumes the script execution speed and continues running the program without trajectory deviation. The running effect is shown in the following figure:



When the robot is paused, click on the programming page. The system will automatically obtain the robot's current position and the robot's entry point by pressing the **"Continue " button.**

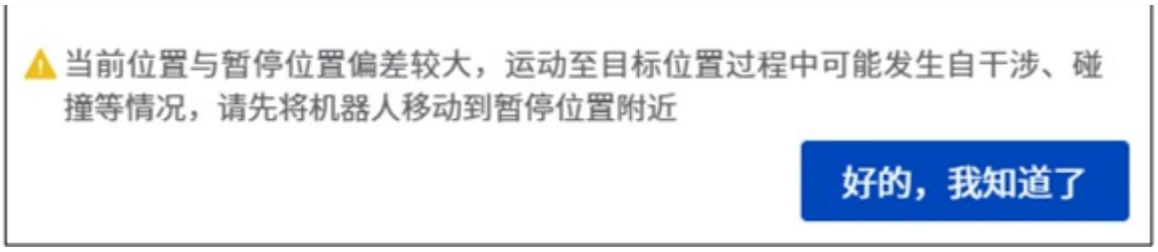
Enters the paused state ? If the robot's current position is consistent with its position when entering the paused state, the robot continues to run the program along the original trajectory.

If the robot's current position deviates from its position when it entered pause mode, the following prompt will appear:



Click **"Continue** ," and the robot will continue moving based on the effect of the trajectory recovery.

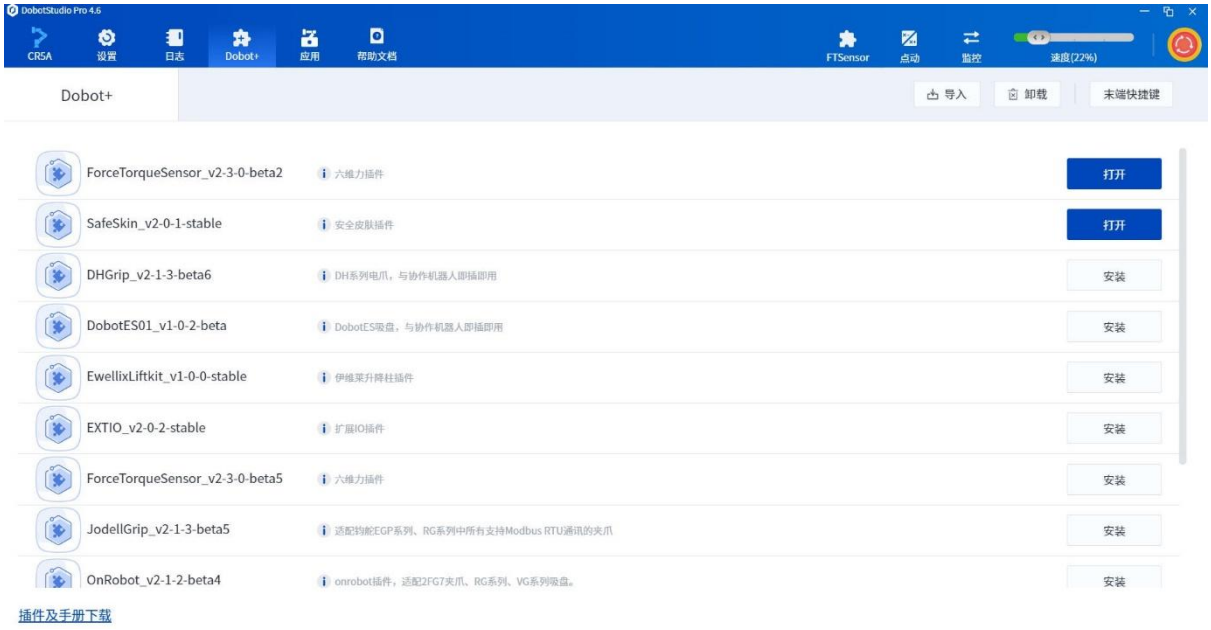
For safety reasons, if the robot's current position deviates significantly from its paused position, and it may experience self- interference or collisions during its movement to the target position, the following warning window will pop up.



At this point , long **press and hold** to move the robot to the pause position.

7 CBSH +

The CBSH + page helps users quickly configure and use CBSH ecosystem accessories, savin g users the trouble of secondary development.



Install button on the right , or by clicking the button in the title bar above. After the document is in use, click the title bar. The **uninstall** button can uninstall the selected plugin.

The plugin is installed , the **install** button will change to an **open** button. Clicking it will open the plugin in a new tab, and multiple plugins can be opened simultaneously. Different plugins have different usage methods, which will not be detailed in this article.

Adding a plugin , both **block programming** and **script programming** will also add plugin-r elated blocks/functions to control ecosystem components during project execution .

Clicking CBSH + **the end-of-line shortcut in the upper right corner of the page** allows you to set the shortcut function for the robot's end-of-line button.

First, select **the plugin list** , then select **shortcut key 1** , **shortcut key 2** , and **long press**the corresponding plugin function. Pressing the end button on the device will execute

the function corresponding to **shortcut key 1 , and pressing the end button again will execute** the function corresponding to **shortcut key 2 .**

Clicking the **"Plugins and Manuals Download" button in the lower left corner of the**

CBSH page will bring up the following prompt box. You can view and download the list

of plugins supported by CBSH and the corresponding user manuals through the

webpage or QR code.

8 monitor

8.1 Control cabinet DI/DO

8.1.1 DI/DO monitoring

This page is used to monitor and configure the status and functions of the DI and DO of the c ontrol cabinet.

The middle area of the page is used to view and set the aliases and status of DI/DO. The control cabinet DI/DO is **a general-purpose I/O** by default , but can be configured as **a system I/O** .

- The functionality of **general-purpose I/O** is user-defined; **aliases** are empty by default. (Click) It is editable, and users can use aliases to annotate the IO function for easy reference during subsequent programming and other operations; after modification, it will be displayed in black text, and clicking on the text will allow for further modification.
- **System IO** refers to DI/DO with specific functions configured on the **IO settings pa ge** below . **Aliases** are displayed in blue text and cannot be modified .

To the right of DI indicates the current state of the corresponding DI; gray indicates OFF,

and green indicates ON.

to the right of the DO indicates the current state of the corresponding DO. Clicking the switch toggles the ON/OFF state, controlling the status of the corresponding DO. **System I/O** cannot be manually switched.

8.1.2 I/O Settings

Clicking the **settings** button at the top of the page will take you to the settings page, where you can configure the system's I/O triggering functions, I/O triggering modes, and types.

Switch to Virtual DI

Click the right side of DI The icon will change After saving, the corresponding DI can be switched to a virtual DI. When the DI is set to a virtual DI... Afterwards, the indicator light on the monitoring page changes to the same switch as DO, and clicking it toggles the virtual DI on/off. By switching the virtual DI, users can simulate the input of an external DI and debug DI-related functions, such as meeting the DI-related judgment conditions during project operation to allow the project to continue running.

illustrate :

The virtual DI is set, it remains in effect. During project execution, reading the corresponding DI via commands will retrieve the virtual value, and the actual value of the DI cannot be obtained . To avoid this, please switch the DI back to a real DI before running the project.

System I/O

Through the drop-down list in the function configuration column. The corresponding function descriptions are as follows.

DI Function	illustrate
Start	When the robot is idle, it will start running the specified project. See the project selection in remote control for details . When the robot pauses, the process (or other form of instruction queue) continues to run.
Stop	Stop the running project (or other form of instruction queue).
Pause	Pause the running project (or other form of instruction queue).
Enable	When the robot is powered on, enable the robot control function.
Enable	When the robot is already enabled, control the robot to de-enable.
Clear alarm	Clear the robot's current alarms.
Enter drag and drop	When the robot is enabled, control the robot to enter drag mode.
Exit drag and drop	When the robot is in drag mode, control the robot to exit drag mode.

DO Function	illustrate

Running status	Output 1 when the robot is running a program, TCP mode instruction queue, or trajectory reproduction; otherwise, output 0. This state indicates whether the robot is running a program and is unrelated to the robot's motion state.
Stopped state	Output 1 when the robot stops, otherwise output 0.
Paused	The robot outputs 1 when it is paused, and 0 otherwise.
Safety origin state	The robot outputs 1 when it is at the safe origin point , and 0 otherwise.
Safe skin pause state	The robot outputs 1 when it is in a paused state triggered by the safety skin, and 0 otherwise.
Idle state	The robot outputs 1 when it is in an idle state (enabled, stopped, and without alarms), and 0 otherwise. The idle state indicates that the robot can accept and execute commands at any time.
On state	The robot outputs 1 when it is powered on, and 0 otherwise.
Enable state	The robot outputs 1 when enabled, otherwise it outputs 0.
Alarm status	The robot outputs 1 if there is an unresolved alarm, otherwise it outputs 0.
Collision state	The robot outputs 1 when it detects a collision, and 0 otherwise.
Drag state	Output 1 when the robot is in drag mode, otherwise output 0.
Low when not running	When the project, TCP mode instruction queue, or trace reproduction is not running, paused, or stopped , it outputs 0. During operation, the corresponding IO status can be set via instructions.
High when not running	When the project, TCP mode instruction queue, or trace reproduction is not running, paused, or stopped , it outputs 1. During operation, the corresponding IO state can be set via instructions.

Low during abnormal shutdown	The robot outputs 0 when it stops abnormally. During operation, the corresponding I/O state can be set via commands. The following situations will cause an abnormal stop: ●Safety functions are disabled (e.g., collision detection, safety walls and safety zones, safety I/ O , etc.). ●The robot is alarming. ●An error occurred during project execution.

The three states— **low when not running** , **high when not running** , **and low when abnormally stopped— are different from the other states. They will only** output the specified state when the conditions are met . In engineering or TCP mode, the state can be freely changed by commands, which helps users quickly control external devices and assist robots with a single DO, simplifying the judgment logic.

illustrate :

●**"Not running** " means the program has not started. When not running, the I/O state is locked and cannot be changed.

●**Abnormal termination** refers to a program stopping due to abnormal situations such as syntax errors, collisions, or emergency stops. I/O states are not locked during abnormal termination.

Case Study :

when **not in operation to illustrate how to use this state: In a glue application application, the user controls the glue** applicator via DO1 during the project . DO1 is 1 when the glue applicator is running, and DO1 is 0 when the glue applicator is stopped. By setting DO1 to **low when not in operation** , the glue applicator can be automatically stopped when the project is not running, paused, or stopped, without affecting the control of glue application via commands during project operation.

Machine designated as **alternatives** cannot be configured on this page; they must be configured first on [the website/platform]. Release the DI in the remote control page. After modifying the configuration, click the **save** button in the upper right corner of the page to complete the configuration.

⚠ Notice :

- All remote trigger sources are active simultaneously. For equipment and production safety, please ensure that the robot can only be started by one control source (control software /DI/Modbus).
- Whether IO triggering takes effect is also affected by manual/automatic mode and IO/Modbus configuration; please refer to the relevant instructions for the corresponding function for details .
- Do not send control signals before the robot has completed its initialization process, otherwise it may cause abnormal robot behavior.

Advanced settings

Trigger mode

This is used to configure how the DI function is triggered. **A rising edge** indicates that the configured function is triggered when DI changes from OFF to ON, **and a falling edge indicates** that the configured function is triggered when DI changes from ON to OFF.

Mode Selection

the CRA/C RAF series , the DO mode needs to be selected according to the actual hardware wiring mode. Please refer to the corresponding CRA hardware user manual for details. After modifying the configuration, click **the save** button to complete the configuration.

> ⚠ **Notice :**
> CBSH Studio must be configured simultaneously. Only after configuring the Pro software parameters and completing the hardware wiring for **the CRA/CRAF series** can the DO mode be successfully configured .

8.1.3 Remote control

Clicking the **remote control** button at the top of the page will take you to the remote control page in **the system settings** , where you can configure the operation of remote control IO monitoring .

8.2 Control cabinet AI/AO

This page is used to monitor and configure the status and mode of the AI and AO of the control cabinet.

> ⓘ **illustrate :**
> Connect to Magician This page is not available with E6 robots.

Analog input/output is used to display the actual values of the control cabinet's AI interface. It supports two detection modes: voltage and current. You can switch between them by clicking the **"Modify "** button . The AO output value can also be manually modified. After setting, you need to click the **"Confirm Modification** " button for the chan ges to take effect.

8.3 Terminal I/O

This page is used to monitor the status and mode of I/O at the robot's end effector.

illustrate :

The image below uses the CRA/CRAF series with a single flight connector as an example. For models with multiple flight connectors (CR20A/CR20AF), multiple tabs will be displayed.

Magician The E6 terminal only has 2 DI and 2 DO channels, lacking RS485 and AI interfaces. The new Nova series terminal only supports RS485 operation and lacks analog input settings.

D I/DO

Click on the DI/DO alias column. Check when you perform the operation .

To the right of DI indicates the current status of the corresponding DI; gray indicates OFF, and green indicates ON.

to the right of DO indicates the current state of the corresponding DO. Clicking the switch will toggle it ON/OFF to control the state of the corresponding DO.

Virtual DI

Click the **settings** button in the upper right corner to configure the virtual DI. When the DI is set to virtual DI, the indicator light will turn into a switch; click to toggle the virtual DI on/off. By switching the virtual DI, users can simulate the input of an external DI device and debug DI-related functions, such as meeting DI-related judgment conditions during project operation to allow the project to continue running.

illustrate :

The virtual DI is set, it remains in effect. During project execution, reading the corresponding DI via commands will retrieve the virtual value, and the actual value of the DI cannot be obtained . To avoid this, please switch the DI back to a real DI before running the project.

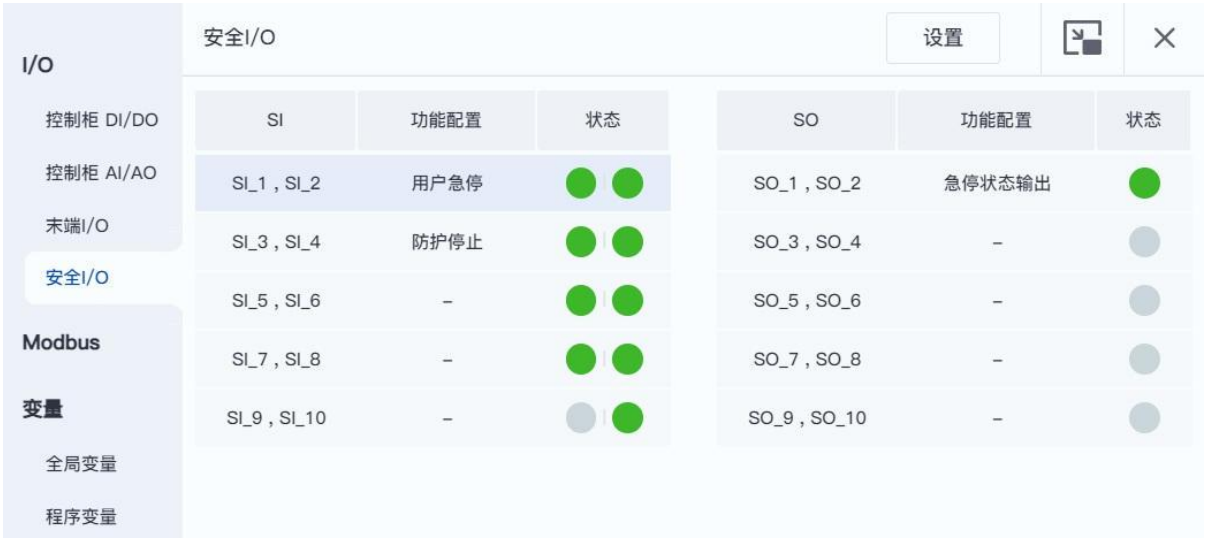
End-point working mode

- When the operating mode is **RS485** , the functions of the terminal IO pins 1 and 2 are 485A and 485B, respectively.
- When the operating mode is **analog input** , the functions of the terminal IO pins 1 and 2 are AI_1 and AI_2. Input values can be monitored in real time on this interface .

illustrate :
When using RS485 terminal tools, the working mode needs to be set to **RS485** .

8.4 Safe I/O

This page is used to view the status of secure I/O and configure secure I/O functions.



The safety I/ O status bar indicate the status of the corresponding I/O, with green indicating a high level and gray indicating a low level.

Security I/O is configured for dual-path. For dual-path SI, the two status lights indicate the status of each path, and the triggering logic varies depending on the configured function; for dual-path SO, the two paths are synchronized, using only one status light.

illustrate :
Magician The E6's safety I/O shares the same terminals as the control cabinet's DI/DO , and can be configured as a single or dual channel as needed . Terminals already configured as system I/O cannot be configured with safety I/O functions, and vice versa.

Click **the settings** button to modify the security I/O configuration. After making changes, click **the save** button to complete the configuration.

Configurable SI functions

Function	illustrate
User emergency stop	The user emergency stop input is an emergency stop interface provided for user use, allowing users to connect external emergency stop devices. The user emergency stop input **defaults to a high-level, normally closed signal input; a low level on any of the inputs triggers the robot** to **enter an emergency stop state . This function will trigger an emergency stop output by default, which may cause emergency stop self -locking in some application scenarios. To avoid this, the configurable** SI interface can be modified in the control software to be **the user emergency stop input (without state output)** , and the corresponding interface can be used as the user emergency stop input. S I_____1 and SI_2. This function is fixed in the configuration.
Protection stopped	The protection stop input is an interface used to connect external protective equipment (such as safety doors, safety light curtains, etc.) . For protection stop **is a high-level normally closed signal input. A low level on any of the inputs will trigger the robot to enter the protection stop state (pause state)** . ●When the protection stop reset input interface is configured, the protection stop input signal and the protection stop reset input must be restored simultaneously before the protection stop state can be released. Then, after confirming continued operation in the host computer, the robot will resume operation. ●If there is not configured , restoring the protection stop input signal will resolve the issue. The protected stop state, the robot resumes operation. SI_3 and SI__4. This function is fixed in the configuration.

Protection Stop Reset	The Protection Stop Reset Input is an interface used to reset the protection stop state. The protection stop reset input **is set to a high-current normallyopen signal input by default. Both rising edges of the signal simultaneously will trigger a reset of the protection stop state** .
Reduced mode	The reduced -mode input is the interface for users to control the robot to enter reduced-mode. It defaults to a high-level, normally-closed signal input. A low-level input on any channel triggers the robot to enter reduced-mode. Returning to a high-level input exits reduced-mode and returns the robot to normal mode. When the robot triggers reduced-mode , the CRA/CRAF's movement speed and safety limits will switch to the settings corresponding to reduced-mode. The global rate of the E6 is limited to 10% and cannot be changed to avoid safety risks caused by high-speed robot operation.

Notice :

Actual use , please try to ensure that the interval between SI signal changes is more than 150ms; otherwise, the SI signal jump may cause abnormal robot operation .

For example, if the protective stop reset input is not configured, the transition of the protective stop input signal (with a change interval of less than 150ms) may cause the robot to fail to resume operation automatically after pausing. This needs to be resolved using any of the following methods:

Using the control software, and then resumed. The protection to stop input signal is triggered again and held for more than 150ms.

Configurable SO functionality

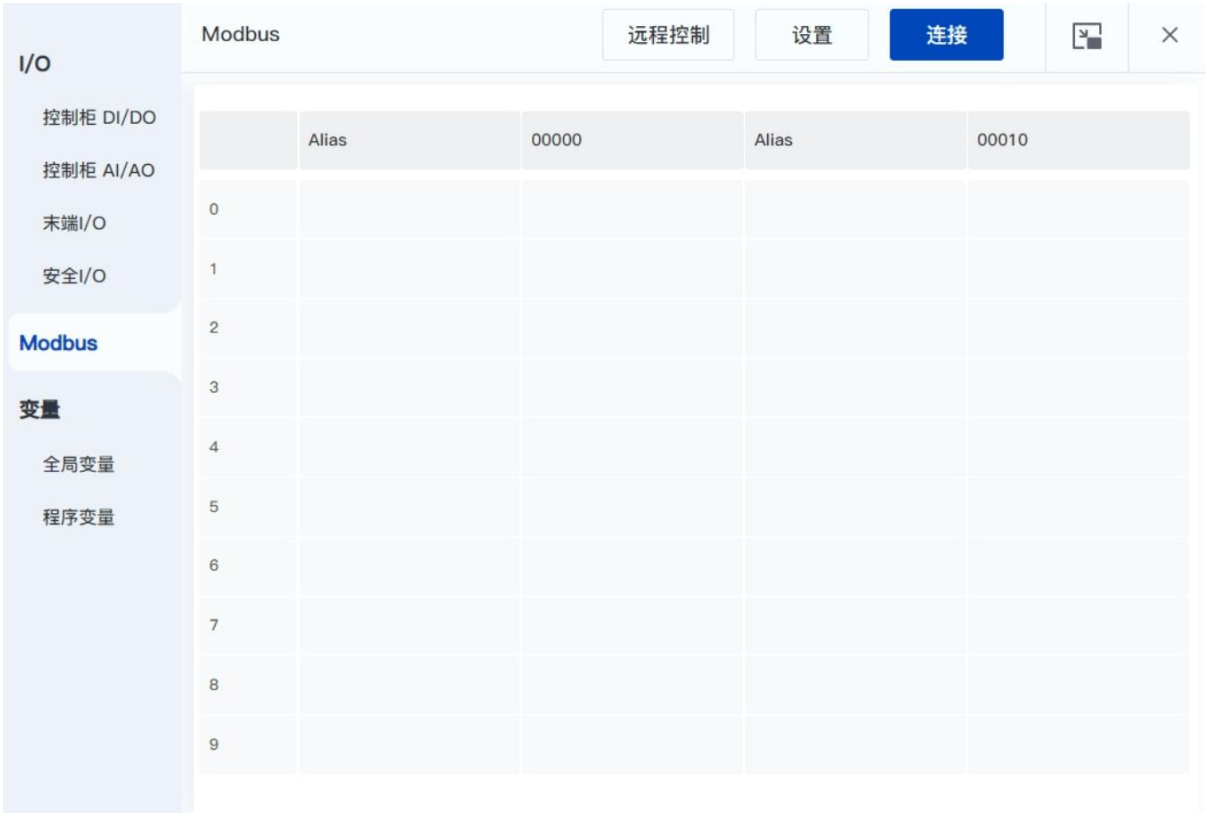
Function	illustrate
Emergency stop output	The robot **is in an emergency stop state, the output voltage is low** ; otherwise, the output voltage is high . This output will be triggered by any sudden stop caused by any source. S O_____1 and SO_2. This function is fixed in the configuration.

Non- stop output	The robot **is in automatic operation mode, it is in a non-stop state and the output voltage is low** ; otherwise, the output voltage is high. This state is whether the robot is performing a task, not whether the joints are moving. For example , during task execution, if the program is waiting for a specified DI to become ON and the robotic arm is not moving, this is a non-stop state, and the output is low. When the task is paused, this is a stop state, and the output is high.
Reduced mode status output	The robot **is in shrink mode, the output voltage is low** ; Otherwise, the output voltage is high .
State output during m otion	If **one or more joints of the robot move more than 1°/s (except in drag mode), it is in motion and the output voltage is low** ; otherwise, the output voltage is high.
Safety origin state out put	The robot **is in a safe origin pose, the output voltage is high** ; o therwise, the output voltage is low . The safe origin can be modified in the security settings.
Protection stop status output	The robot **is in a protected stop state, the output voltage is lo w** ; otherwise, the output voltage is high .
System emergency stop status output	the robot **is in a system emergency stop state, the output voltage is low** ; otherwise, the output voltage is high . The system emergency stop is triggered by the emergency stop button or software emergency stop.
User emergency stop status output	**When** the robot is in an emergency stop state, the output voltage is low ; otherwise, the output voltage is high . User emergency stop is an emergency stop triggered by safe I/ O.

8.5 Modbus

8.5.1 Modbus monitoring

This page is used by the host computer as a Modbus master to connect to the built-in Modbus slave of the robot controller to view and modify register values, as well as to configure the remote Modbus control function when the controller is a slave.



Click **the link** at the top of the page to set the slave station to connect to.

- **Slave IP** : The address of the Modbus device. When connecting to the Modbus slave device that comes with the control cabinet, enter the control cabinet's IP address, for example, 192.168.200.1 .
- **Port** : The port number for Modbus communication. Enter 502 when connecting to the M odbus slave unit provided with the control cabinet.
- **Slave ID** : The ID of the slave device.
- **Function code** : Select the function type of the slave device.
- **Address/ Quantity** : The address and quantity of the registers. When connecting to the Modbus slave station provided with the control cabinet, please refer to Appendix A for the Modbus register definitions .
- **Scan cycle : The time interval at which the robotic arm scans the slave station. After successful connection, a table in the middle of the page will display** the aliases and values for each slave station address . Double-click (PC) or single-click (mobile) the cell containing the alias to modify the alias. The register type is coil. When using a register or a holding register, double-click (on PC) or click (on mobile) the cell containing the register value to modify the value.

8.5.2 Modbus settings

Click the **settings button at the top of the page to enter the settings page, where you can set the trigger mode of the coil register and view** the address configuration information corresponding to the remote control function

The trigger mode is used to set how the function of the coil register is triggered. **The rising edge** indicates that the configured function is triggered when the coil register changes from 0 to 1 , **and the falling edge** indicates that the configured function is triggered when the coil register changes from 1 to 0.

The coil register and **contact register** can only be viewed and cannot be modified. The corres ponding functions are described below.

Coil Register Function	illustrate
Start	When the robotic arm is idle, it will start running the specified project. See the project selection in remote control for details . When the robotic arm is paused, the process (or other form of instruction queue) continues to run.
Stop	Stop the running project (or other form of instruction queue).
Pause	Pause the running project (or other form of instruction queue).
Enable	When the robot is powered on, enable the robot control function.
Enable	When the robot is already enabled, control the robot to de-enable.
Clear alarm	Clear the robot's current alarms.
Enter drag and drop	When the robot is enabled, control the robot to enter drag mode.
Exit drag and drop	When the robot is in drag mode, control the robot to exit drag mode.

Contact register function	illustrate
Running status	Output 1 when the robot is running a program, TCP mode instruction queue, or trajectory reproduction; otherwise, output 0. This state indicates whether the robot is running a program and is unrelated to the robot's motion state.
Stopped state	Output 1 when the robot stops, otherwise output 0.
Paused	The robot outputs 1 when it is paused, and 0 otherwise.
Safety origin state	The robot outputs 1 when it is at the safe origin , and 0 otherwise.
Safe skin pause state	The robot outputs 1 when it is in a paused state triggered by the safety skin, and 0 otherwise.
Idle state	The robot outputs 1 when it is in an idle state (enabled, stopped, and without alarms), and 0 otherwise . The idle state indicates that the robot can accept and execute commands at any time.

On state	The robot outputs 1 when it starts to power on (this does not indicate that power-on is complete); otherwise, it outputs 0.
Enable state	The robot outputs 1 when enabled, otherwise it outputs 0.
Alarm status	The robot outputs 1 if there is an unresolved alarm, otherwise it outputs 0.
Collision state	The robot outputs 1 when it detects a collision, and 0 otherwise.
Drag state	Output 1 when the robot is in drag mode, otherwise output 0.
Recovery mode status	The robot outputs 1 when it is in recovery mode , and 0 otherwise.

Modifying the configuration, click the **save** button to complete the configuration.

Notice :

- All remote trigger sources are active simultaneously. For equipment and production safety, please ensure that the robot can only be started by one control source (control software /DI/Modbus).
- Modbus remote control signals may experience some delay due to network interference. Please determine whether the delay is acceptable based on on-site testing .
- Modbus triggering takes effect is also affected by manual/automatic mode and IO/Modbus configuration; please refer to the relevant instructions for the corresponding function for details .
- Do not send control signals before the robot has completed its initialization process, otherwise it may cause abnormal robot behavior.

8.5.3 Remote control

Clicking the **remote control** button at the top of the page will take you to the remote control page in **the system settings** , where you can
configure remote control of Modbus monitored projects .

8.6 global variables

This page is used to set global variables. Global variables are persistently stored in the controller (and can be retained even after a power outage) and can be accessed by multiple projects .
Setting global variables, they can be called using blocks related to global variables in **block programming , and can** be called directly by variable name in **script programming .**

Click The **"Add** " button allows you to add new global variables. Select the variable and Click and modify the properties of a variable The button can **delete** the selected variable.

The following types of global variables are supported :

- Numeric : Value. Supports setting a value range; values exceeding the range will not be saved. If a numeric variable is assigned a value exceeding the range in the project , the robot will report an error and stop. Only administrators can modify the value range (this permission cannot be assigned).
- Bool : Boolean value (true/false)
- String: String. Strings entered on this page do not need to be enclosed in double quotes, but when modifying the value of a global variable of type string in the project, t he string needs to be enclosed in double quotes.
- Table : A table (containing arrays). It can be set to points or a custom format.
  - W h e n setting to **point** format, move the robot to the point you want to save, and then click " **Get Point**" .

  - W h e n setting a **custom** format, you need to enter the value of the variable. The variable value format entered and displayed on the global variables page must meet the restrictions of JSON data format, which differs from the data form at used in the project. The specific rules are as follows:
    - For arrays , you need to use [] instead of {} in the global variable interface. For example, if the array in the project is formatted as {1,2,3}, the global variable interface should input and display it in the format [1,2,3].
    - For tables in the format {key = value}, you need to change it to {"key": value} in the global variable interface. For example, in the project, the format is {a =1, The array b="test"} is used as the global variable in the interface, with {" a":1, Input and display in the format “b”:"test"} .

After setting up, click **"Add"** to add a global variable. **Keep it globally :**

- When checked , any modifications to this variable will be saved, and the modified value will be saved even after exiting the script or restarting after a power outage.
- When not checked , modifications to this variable only take effect while the script is running; exiting the script will restore it to its initial value. Furthermore, the real-time value of this variable will not be visible in the global variable monitoring list while the script is running. It can only be viewed through program variables .

Usage restrictions:

- Global variables only support Chinese and English.
- When using **global variables of type ` table`** , please avoid the following situations, o therwise the robot will report an error and stop the project . The `table` in the following text..._____1 and table_Both_2 are global variables of type table that have already been added.
  - **Nested variable values**

```
-- Correct usage: Assign constants or other variables to members of the table.
t able__1[ 1 ] = { 1 , 2 , 3 }
t able__1[ 2 ] = table____2
-- Incorrect usage: Assigning a table member to the table itself.
t able__1[ 1 ] = table____1 - - ta ble    ___The member of 1 is assigned the value table._1 itself will cause an error.
```

8.7 Program variables

This page is used to set the variables that need to be monitored when running the project. By variable name, you can monitor the type and value of the specified variable, and you can also modify the variable name of the monitored variable.

Click The "Add " button allows you to add new The Edit button allows you to modify the variable's properties.Select the variable and click... The button can delete the selected variable.

Variable names can be entered as object key values:

- For example, to monitor the joint coordinates of point P1, add the variable: P1.joint
- For example, if you want to monitor the X-axis of point P1, add the variable: P1.pose[1].

After adding the variable, the variable list will display the added variable.

Program variable monitoring

Program variable monitoring starts when the script runs, polling every 1 second; polling stops when the script is paused/stopped. Added program variables, if they do not have values during program execution, will display both their type and value. '-'

Program variable types

Program variables are categorized into three types based on their source: local, up value, and global. When variables have the same name, the former has the highest priority, and therefore the variable displayed and modified will be the one with the highest priority.

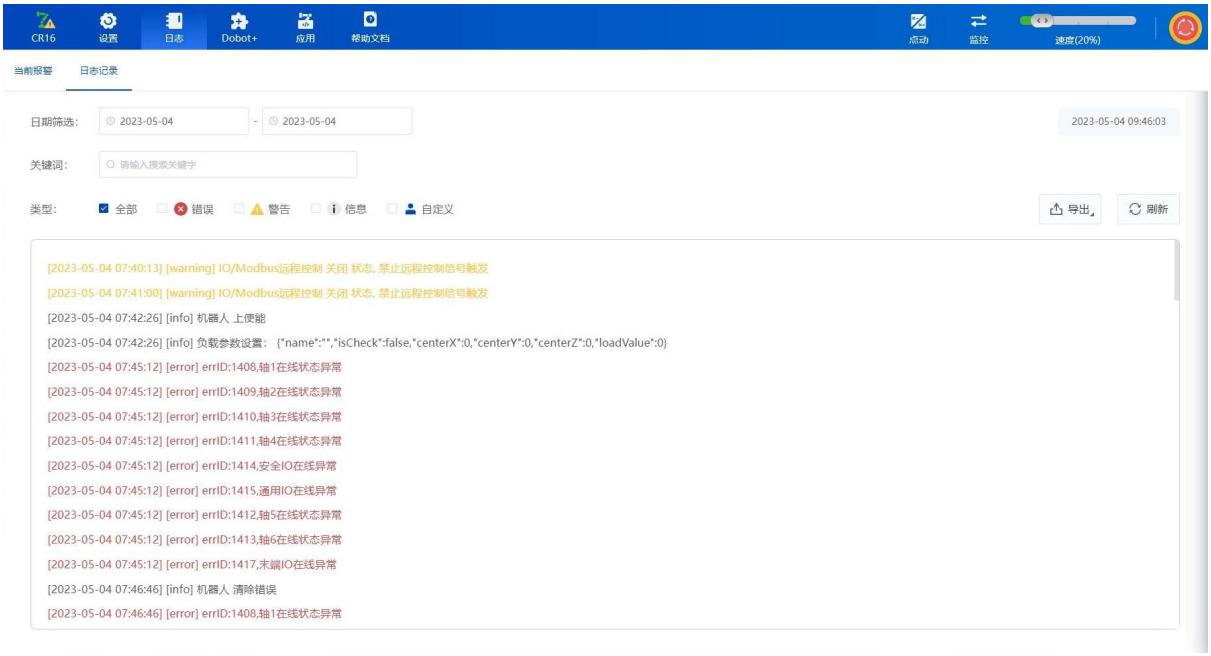
The native Lua variable types that can be monitored and viewed include:

- nil: Represents empty (without any valid value). For example, printing an unassigned variable will output nil. value.
- boolean: A boolean value, only true (correct). and `false` has two possible values. (Lua) Bundle false and nil If we consider it false, everything else is true, including the number 0. true.
- Number : A numeric value, a double-precision floating-point number that supports various operations.
- string : A string consisting of numbers, letters, and underscores.
- table : a table (containing arrays). The table type supports expanding to view the content and allows modification of its child elements.

Usage restrictions

- When the script is running, paused, or stopped, it supports adding, deleting, and modifying program variables.
- The program can add a maximum of 20 variables, and the variable name is limited to a maximum length of 256 bytes.
- Program variables only support Chinese and English.

9Log



The log page displays the robot's operational logs and supports filtering by date, keywords, and log type. The meanings of the log types are as Follows:

- Error log: Alarm information related to the software performing incorrect actions or the robot exhibiting an alarm state.
- Warning Log : Warning messages related to abnormal software actions or abnormal robot states. Abnormal states do not affect the execution of subsequent actions .
- Information log: Information recorded when the robot's state changes.
- Custom logs: Log information output by the user using the `Log(value) command.`

Log export

Export to USB : Exports logs that meet the filter criteria to a storage device connected to the control cabinet's USB interface. **Export to Local** : Exports logs that meet the filter criteria to the local computer.

Export Manufacturer Logs to USB : This function is only supported on PC and exports all controller logs via USB. **Export Manufacturer Logs to Local** : Exports all controller logs to your local computer.

illustrate :

If the storage device connected to the USB port contains multiple partitions, the logs will be exported to the first partition. (Some storage devices...) (For example, a USB flash drive used as a boot disk) If the first partition is a hidden partition, the exported logs cannot be directly viewed in Windows.
the manufacturer logs are successfully exported, they will be saved in a folder named "logs" in the root directory of the USB drive. When exporting the manufacturer logs again , the new "logs" folder will overwrite the original folder.
Avoid unplugging the USB drive during the log export process, as this may corrupt the files on the USB drive. Do not operate the robot or close the software while the logs are being exported to your local computer.

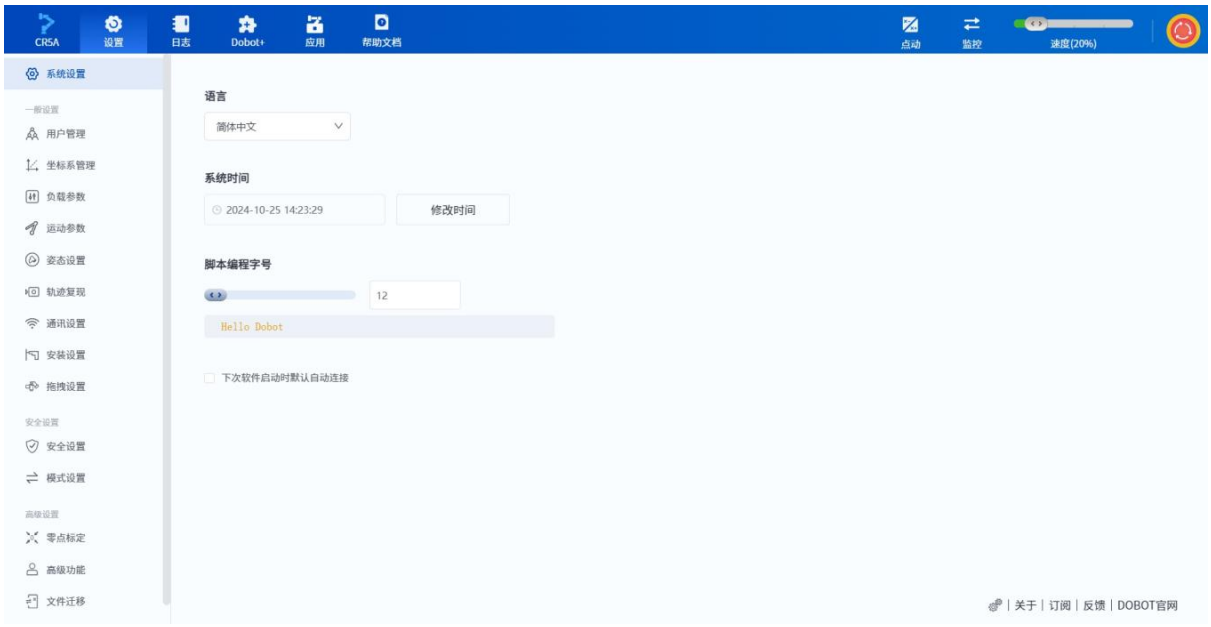
Some scenarios (such as when the control software is connected to the robotic arm while it is running automatically), the logs will not refresh automatically, and you need to manually click **refresh** to get the latest log records.

10 set up

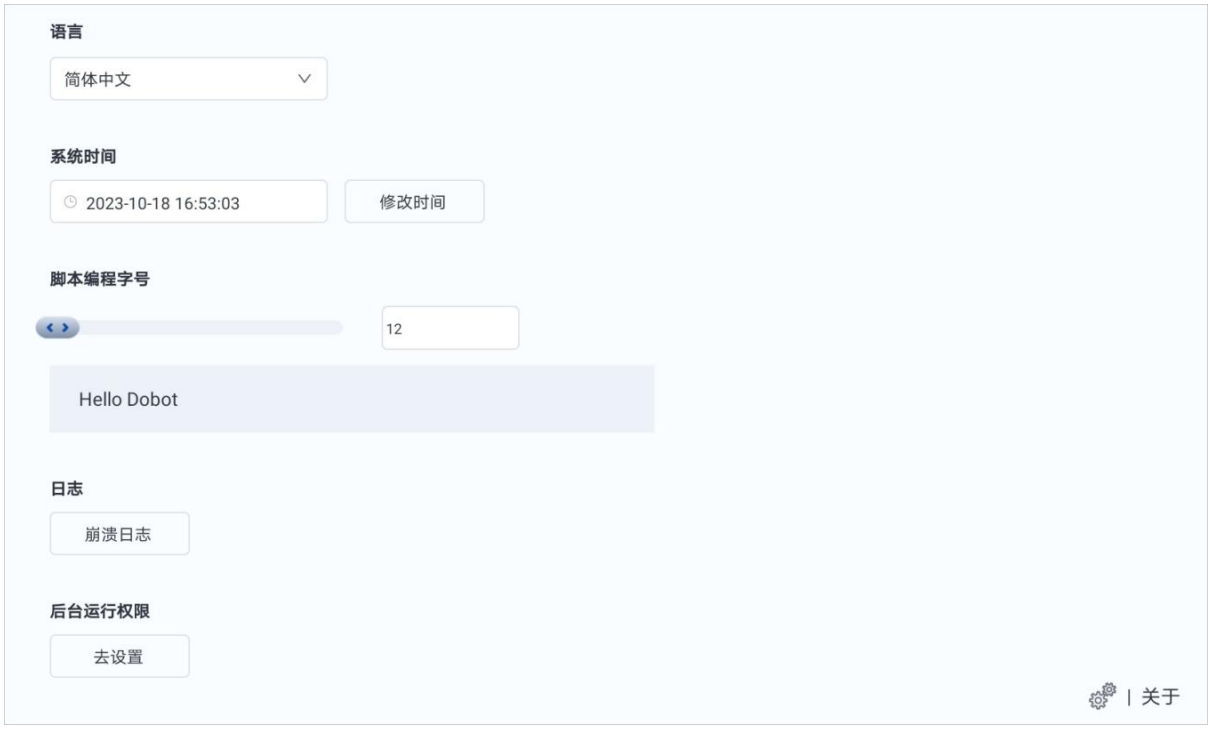
10.1 System Settings

The system settings page is used to modify the software interface display language, system time, script programming font size, etc.

PC end



Mobile



Language : CBSH Studio settings The Pro interface display language can also be changed when the robot is not connected.

> illustrate :
> If you are unable to set your desired language, please contact technical support.

System Time : Displays the controller's current system time. This can be modified in default or manual mode when no project is running. If the current device 's system time differs from the controller's system time, a prompt will appear on the interface, suggesting that it be corrected to match.

Scripting Font Size : Sets the font size of the scripting code area. The default is 14, and the value range is [12, ...]. 50].

Logs : A mobile feature that uploads crash logs. Requires an internet connection and can be used even when not connected to the robot.

Next software launch : When checked, the software will attempt to automatically connect to the currently connected robotic arm on the next software launch. PC only .
The icons and text in the lower right corner of the interface are clickable, and their functions are as follows:


To use the manufacturer's functions, you need to enter the manufacturer's password. Please use it under the guidance of technical support.

About : View the software CBSH Studio Pro components and startup disclaimer. The following features are only supported on PC.

Subscribe : Subscribe to the latest news on software updates and products.

Feedback : Please provide feedback on any issues encountered while using the software.

CBSH official website : Click to open browser access CBSH official website .

10.2 User Management

When a user logs in as an administrator, they can manage the permissions and passwords of functions on this page, and can also add or delete custom functions.

Set default function

In the upper left corner of the page , you can set a default role (not administrator). This role is the one that automatically logs in each time you connect to the robot. If the defaultr ole is password enabled, you must enter the password or select another role to log in each time you connect to the robot before you can perform other operations.

Manage custom functions

Click ┼**"Add Function"** to add up to two custom functions.
Custom roles can be renamed and deleted; other operations are the same as default roles. Please refer to the following text .

Set functional password

The functional password . It supports a combination of English letters and numbers of no more than 20 characters. Please set it according to your actual needs .

The administrator password must be enabled. **The default password is 888888.** It can be c hanged, but the old password must be entered when changing it.

Other functions do not enable passwords by default, **and the password enable/disable** switch is **OFF by default** . Clicking the switch will bring up a password setting window . After setting the password, the switch will become **ON** .

Setting a password , clicking **"Set Password" again will bring up the same password setti ng window. You can reset** the password for the corresponding function by entering the new password directly, without needing the old password .

Turning off **the password enable** /disable switch will clear the current password set for the

corresponding function, and you will need to reset the password the next time you turn it on.

Set functional authority

Clicking on **permission assignment** will take you to the following page (custom functions will only be displayed after they have been added).

All functions can be modified. Clicking " **Restore Default Settings"** will restore the permission assignment status to the default values. After making changes , click **"Save"** to apply the settings.

Default Functional Permissions

The default function permissions are shown in the table below, among which the permissions for **opening/running/stopping the project** function cannot be modified.

Category	Function	Administrator	Technician	Operator
Basic operations	Open /Run/Stop Project	√	√	√
	Enable , clear alarm, switch between manual and automatic modes , adjust speed ratio	√	√	√
	Robot jogging	√	√	√
	Log viewing/export	√	√	√
	Online /TCP mode switching, enabling/disabling IO /Modbus configuration	√	√	X

	IO (including secure IO) configuration	√	√	X
	Modbus configuration	√	√	X
	Global variable settings	√	√	X
	CBSH + Plugin, End Button Settings	√	√	X
	Engineering and process management	√	√	X
	Point list operation	√	√	X
	System time settings	√	X	X
	Coordinate system management	√	√	X
General Settings	Load parameter settings	√	√	X
	Button settings	√	√	X
	Track recording and reproduction	√	√	X
	Communication settings	√	√	X
	power supply voltage	√	√	X
Advanced settings	Packaging /Zero-point attitude setting	√	X	X
	Exercise parameter settings	√	X	X
	Installation angle setting	√	X	X
	Drag and drop settings	√	X	X
	Security Settings	√	X	X
	Manual /Automatic Mode Settings	√	X	X
	Zero point calibration	√	X	X
	Advanced feature settings	√	X	X

Batch operations

Click for **batch operations > The "Export Configuration" option** allows you to export the current user's permission configuration to a file. Click **"Batch Operation" to perform t his operation. > Importing configurations can** import user permission configurations from a file .

10.3 Coordinate system management

10.3.1 User coordinate system

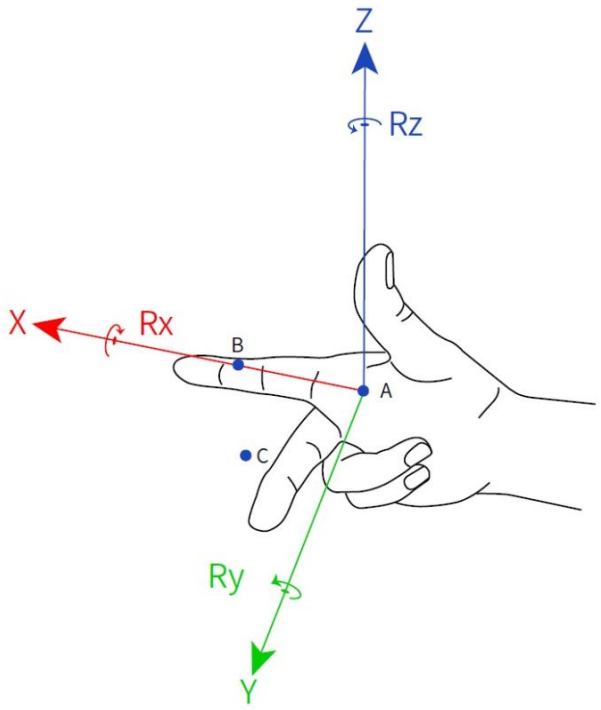
When the position of the workpiece changes or the robot arm's operating program needs to be reused in multiple similar machining systems, it is necessary to set up a user coordinate system so that all paths are updated synchronously with the user coordinates, which greatly simplifies the teaching programming.

The current system supports a maximum of 51 user coordinate systems [0, 50]. User coordinate system **0** is the base coordinate system (see the hardware user manual of each robot for details) and cannot be changed.

The user coordinate system can be customized. The coordinate values of the same robot pose will differ in different user coordinate systems. The values in the user coordinate system represent the offset and rotation angle of that user coordinate system relative to user coordinate system 0.

> ⚠ **Notice :**
> Establishing the user coordinate system, ensure that the reference coordinate system when obtaining the point location is the user coordinate system **0** .

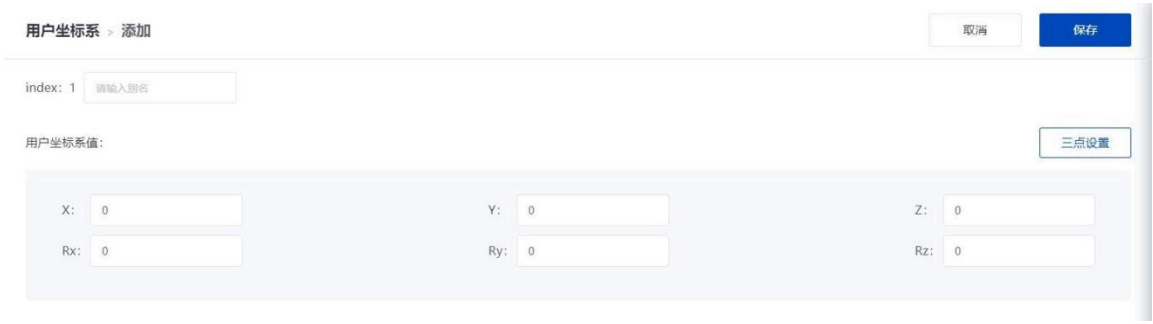
User coordinate system is generated using a three-point teaching method: move the robotic arm to any three points **A** , **B** , and **C.** Point **A** is used as the origin .
Point A and point **B** are connected by a line to determine the positive direction of the X-axis of the user coordinate system. Point **C is perpendicular to the X-axis to determine the positive direction of the Y-axis** . The Z-axis direction is determined by the right-hand rule .



Create user coordinate system

1. Click on the user coordinate system page **Add it** , as shown in the image below.

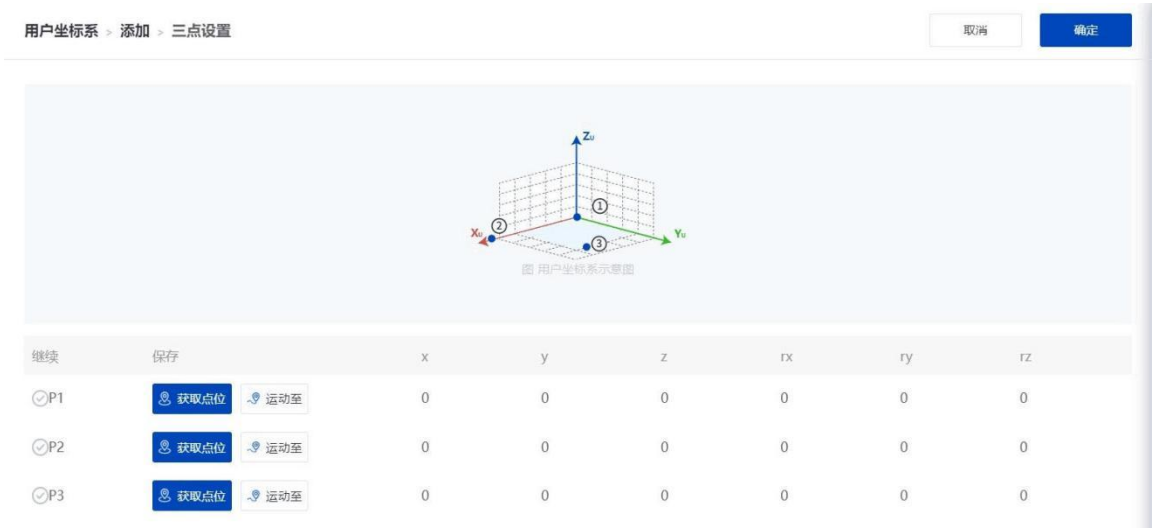
2. On the Add User Coordinate System page, click **Three Points Settings** .



illustrate :

Users can also directly modify the values of X, Y, Z, Rx, Ry, and Rz, and click **save** . X, Y, and Z represent the position of the origin of the user coordinate system in the base coordinate system, and Rx, Ry, and Rz represent the angles of rotation of the user coordinate system around the base coordinate system in the order X->Y->Z .

3. Referring to the diagram, control the robotic arm to move to the corresponding point and then click. **Obtain the location** .



illustrate :

Long press **The movement** allows the robotic arm to move to the acquired point.

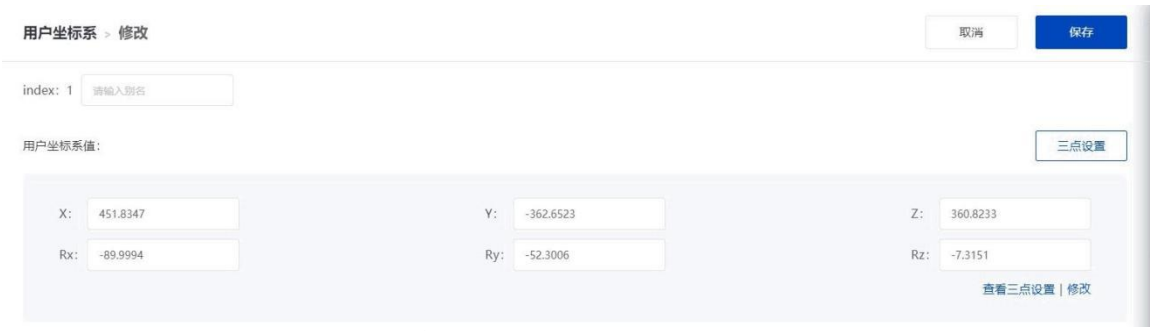
4. Click **OK** to return to the Add User Coordinate System page. The coordinate system values will be updated to the calibrated values. You can view/modify the three-point settings for generating this coordinate system or manually modify the coordinate system values. See the steps below for modifying the user coordinate system.

5. Click **Save** to add this coordinate system to the User Coordinate Series Table.

Modify user coordinate system

1. After selecting the coordinate system on the user coordinate system page, click **The modifications** are shown in the image below.

2. If the selected user coordinate system values are manually entered and saved, the appearance of the modification page is the same as the add page ; you can directly modify them or click **the three-point settings** for calibration. If the selected user coordinate system values are generated by the three-point settings , the UI of the modification page is shown in the following figure.



3. Clicking the **"View Three-Point Settings" button in the lower right corner of the coordinate system value** allows you to see the three-point settings that generated the coordinate system value and re-acquire the points that need to be modified .
4. Click **"Modify"** to directly modify the coordinate system values.

> **illustrate :**
> Manual modification, it is no longer possible to view the three-point settingsc orresponding to the coordinate system values before the modification.

5. Click **Save** to update this coordinate system in the user coordinate system list.

Copy user coordinate system

After selecting the coordinate system on the user coordinate system page, click **Copying** creates a new coordinate system that is identical to the selected coordinate system.

Clear user coordinate system

After selecting the coordinate system on the user coordinate system page, click **Clear** and confirm to remove the selected coordinate system. The cleared coordinate system is shown below.
It still occupies its ID, only the coordinate system data is cleared (e.g., coordinate system 2 in the figure below), and will report an error when it is called.

Clicking **"Hide Empty Coordinate System"** will hide the cleared coordinate system from the coordinate system list. Then the button will change **to "Show Empty Coordinate System"** . Clicking it will restore the display of the empty coordinate system.

Empty coordinate system can be modified, and after modification, it will be reassigned.

10.3.2 Tool coordinate system

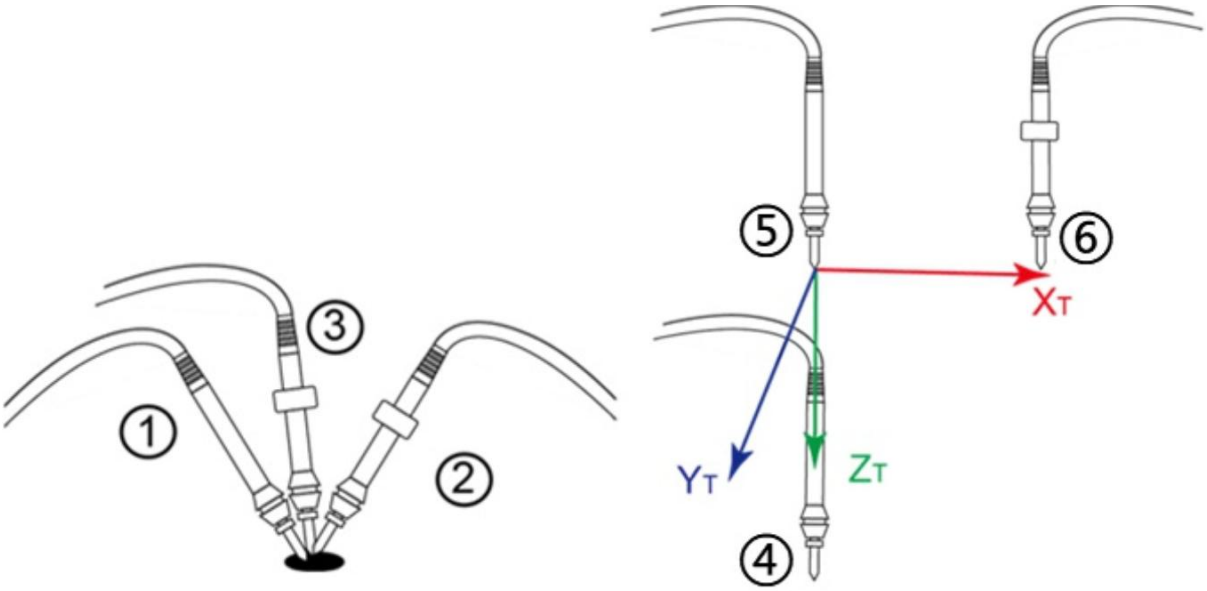
Once tools (such as welding guns, nozzles, fixtures, etc.) are installed at the end effector of a robotic arm , a tool coordinate system needs to be set up for programming and robotic arm operation. For example, when using multiple fixtures to simultaneously move multiple workpieces, each fixture can be set to an independent tool coordinate system to improve handling efficiency.

The current system supports a maximum of 51 tool coordinate systems [0, 50]. Tool coordinate system **0** is the flange coordinate system (see the hardware user manual of each robot for details), which cannot be changed when no tool is used.

The tool coordinate system is based on TCP (Tool) Center Point is generally set as the point of application of the tool, such as the center of a suction cup or the welding torch. A coordinate system is established with the tool head as the origin to represent the tool's position and orientation. The values in the tool coordinate system represent the offset and rotation angle of the tool coordinate system relative to the tool coordinate system 0.

> ⚠ **Notice :**
> Establishing the tool coordinate system, ensure that the reference coordinate system when obtaining the point is the tool coordinate system **0** .

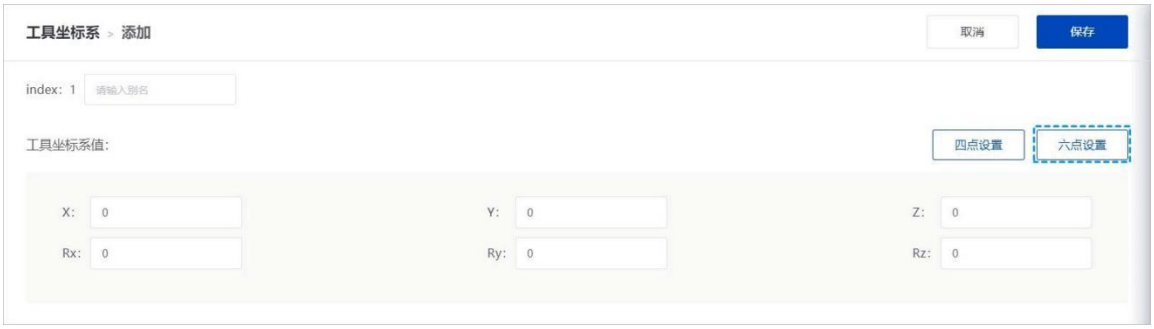
For a six-axis tool is generated using the six-point teaching method "TCP+ZX": After the tool is installed at the end effector of the robotic arm, adjust the tool's position. The tool's position offset is obtained by aligning the TCP with the same point (reference point) in space in three different directions (①②③). Then , the tool's attitude offset is obtained based on three other points (④⑤⑥), where ④ aligns with the same point as ①②③.



Create tool coordinate system

1. Click on the Tools Coordinate System page ＋**Add it** , as shown in the image below.

2. In the Add Tool Coordinate System page, click **the six-point setting in the upper right corner of the coordinate system value** .



illustrate :

Users can also directly modify the values of X, Y, Z, Rx, Ry, and Rz, and then cli ck **save** . The operation method for **four-point settings is similar to that** for **six-p oint settings** , and will not be repeated in this article.

3. Referring to the diagram, control the robotic arm to move to the corresponding point and then click. **Obtain the location** .

illustrate :

Long press **The movement** allows the robotic arm to move to the acquired point.

4. Click **OK** to return to the Add Tool Coordinate System page. The coordinate system values will be updated to the calibrated values. You can view/modify the three-point settings for generating this coordinate system or manually modify the coordinate system values. See the steps below for modifying the tool coordinate system.
5. Click **Save** to add this coordinate system to the Tools Coordinate Series Table.

Modify tool coordinate system

1. After selecting the coordinate system on the Tools Coordinate System page, click **The modifications** are shown in the image below.

2. If the coordinate system values of the selected tool's coordinate system are manually entered and saved, the appearance of the modified page is the same as the added page. You can directly modify them or click on **four-point settings/six-point settings.**
   For calibration. If the coordinate system values of the selected tool's coordinate system are generated by four/ six-point settings, the appearance of the modified page is as follows. **Note that** the text in the six-point settings will change depending on the generation method.

3. Clicking **"View Six-Point Settings"** will show the six-point settings used to generate th e coordinate system values. You can then re-acquire the points that need modification. The same applies to the four-point settings .

4. Click **"Modify"** to directly modify the coordinate system values. Please note that after manually modifying the coordinate system values, you will no longer be able to view the original six/four point settings.

5. Click **Save** to update this coordinate system in the tool coordinate system list.

Coordinate system of copy tool

After selecting the coordinate system on the Tools Coordinate System page, click **Copying** creates a new coordinate system that is identical to the selected coordinate system.

Clear tool coordinate system

After selecting the coordinate system on the Tools Coordinate System page, click **Clear** and confirm to remove the selected coordinate system. The cleared coordinate system is shown below.
It still occupies its ID, only the coordinate system data is cleared (e.g., coordinate system 3 in the figure below), and will report an error when it is called.

Checking **the "Hide empty coordinate system" option** will prevent the cleared coordinate system from being displayed in the coordinate system list. Then the button will change to **"Show empty coordinate system"** . Clicking it will restore the display of the empty coordinate system.

Empty coordinate system can be modified, and after modification, it will be reassigned.

10.4 Load parameters

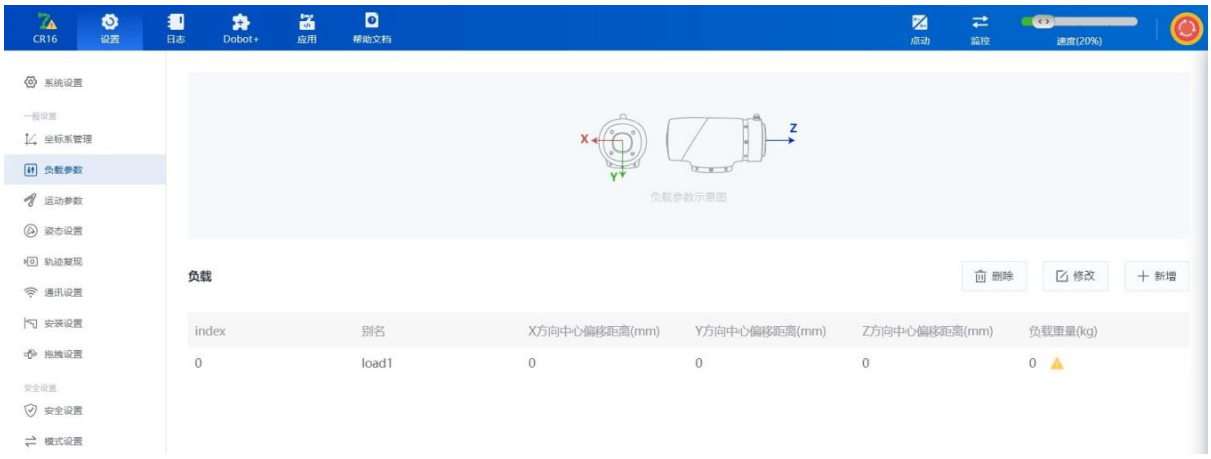
The load parameters are the center of mass and weight parameters of the robot end effector load (including fixtures). Please set them according to the actual load.

illustrate :

This interface is only for modifying load parameter groups; changes will not take effecti mmediately. To make the modified parameters take effect, please select the corresponding parameter group in the load settings page that pops up when enabling t he robot .

Notice :

Load settings can lead to decreased robot performance and may cause collision detection errors or uncontrollable movement of the robot body during dragging.



Click **Adding** a new set of load parameters; after selecting a parameter group, click Group, click **The delete function** can remove the selected parameter group.

Parameter group aliases can be modified; they are used to reference parameter groups during both enabling and programming. There are two methods for setting load parameters: load identification and manual modification.

Load Identification

The robot is enabled, without alarms, and not moving, it can automatically identify the parameters of the robot's current load.

illustrate :

- Performing load identification, please ensure that the robot's installation angle is set correctly.
- Magician E6 does not support load identification.

1. Click **Load Identification** to open the Automatic Identification window.
2. The system will automatically generate 7 default points. Selecting a point will display the corresponding pose diagram. Long press... **Moving to a position** allows the robot to move to the corresponding point.
3. If the robot cannot move to a default point due to obstacles or other reasons, you can refer to the diagram of that point and jog the robot to another point that meets the requirements , then **overwrite** the default point. Clicking **"Restore Default Point"** will restore the overwritten point to the default point.
4. With a **single click, the robotic arm will automatically identify** and confirm each

location, and then proceed to each location sequentially for identification.

- If the identification is successful, the software will return to the load addition/modification interface, and the load parameters will be updated to the identified parameters. Please follow the instructions. Confirm the validity of these parameters based on the actual load conditions. If they are not valid, please re-identify the load or manually modify the parameters .
- If the identification fails, an error message will pop up and you will remain on the automatic identification page. Please refer to the diagram of each point to confirm whether the point meets the requirements. After modifying the points that do not meet the requirements, try **automatic identification again with one click** .

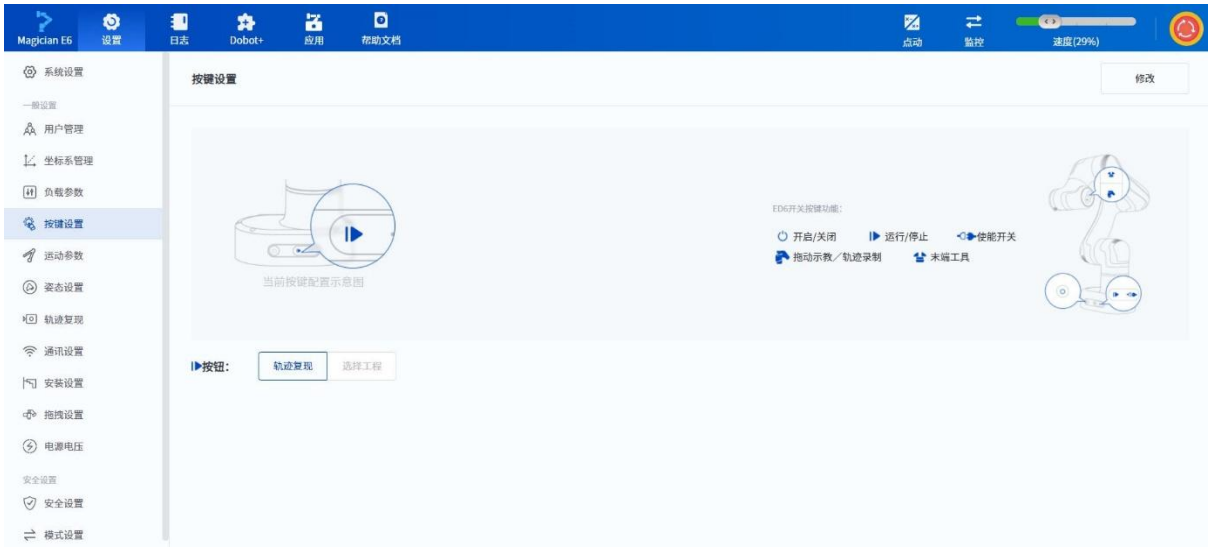
Manual modification

Manually enter the following load parameters:

- **X/Y/Z direction center offset distance** : the offset distance of the end load centroid in each direction. See the schematic diagram on the interface for each coordinate axis direction.
- **End-effector load** : The sum of the end-effector weight and the workpiece weight must not exceed the robot's maximum allowable load. After modifying the settings, click the **save** button to save the settings.

10.5 Key settings (Magician) E6)

The robotic arm type is Magician In E6, the function of the base run/stop button can be set.



Clicking the **"Modify " button** in the upper right corner allows you to modify the function of th e run/stop button.

- The default function of this button is track playback, which reproduces the most recently recorded track;
- If you want to set the function to run a project, you need to select the project you want to run.

Regardless of what function the button is set to, pressing this button during robotic arm trajectory reproduction or operation will stop the operation.

10.6 Motion parameters

The robot's motion parameters are pre-set for optimal operation under normal conditions at the factory; modification is not recommended unless there are specific requirements. If the user finds the robot's speed too high, the speed can be reduced as needed. For speed increases, please contact technical support for assistance.

Case .

CRA/CRAF/New Nova Series and Magician The E6 page is different.

10.6.1 Motion parameters (CRA)

Reproduction parameters

Reproduction parameters are for robot **operation engineering** or **TCP**_Motion parameters when using **IP motion commands .**

Torque Constraint

This feature is crucial for ensuring the reliable operation of the robot, and it is highly recommended to enable it. Once enabled, the robot's algorithm will adjust the acceleration and jerk based on actual operating conditions to prevent the robot from exceeding torque limits and triggering alarms.

Reproduction speed

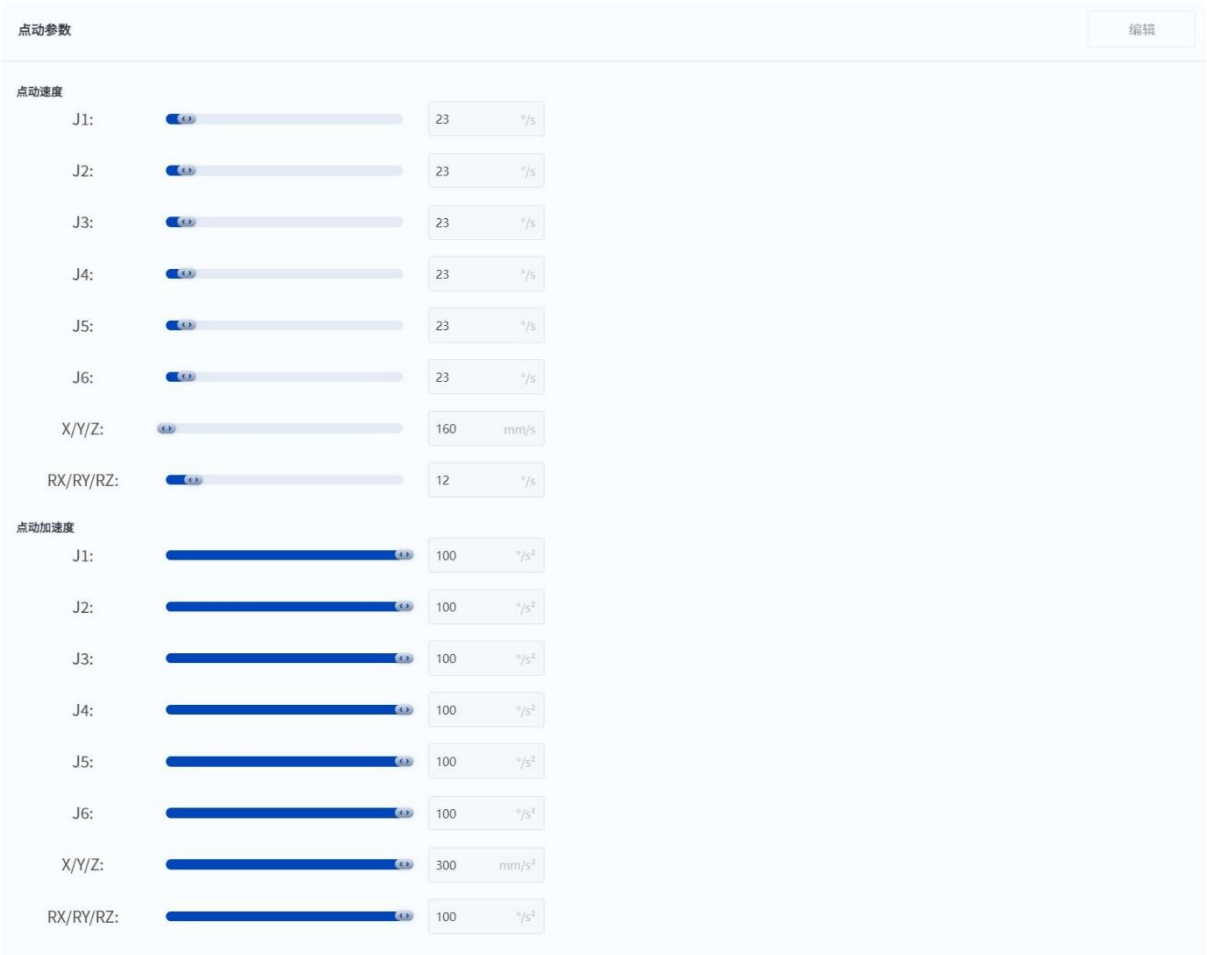
To set the maximum movement speed of each joint of the robot when reproducing motion, you need to set the maximum speed separately for normal mode and reduced mode. The maximum speed in reduced mode cannot exceed the maximum speed in normal mode.
This page only allows you to set joint velocities. The maximum TCP velocity in

Cartesian coordinates needs to be set on the **security limits** page. **Reproduce**

Acceleration/Reproduce jerk

To set the maximum values for the acceleration and jerk of the reproduced motion, you need t o set the values for joint motion and Cartesian motion separately.

Jog parameters



The jogging parameters are the motion parameters of the robot **during jogging/inching** and **when it reaches its destination. Jogging speed**

To set the maximum speed of the jog motion, you need to set the joint speed and Cartesian s peed separately.

Jog acceleration

To set the maximum acceleration for jogging motion, you need to set the joint acceleration and Cartesian acceleration separately.

Factors affecting rhythm

- In practice , the robot's maximum motion speed is determined by both the maximum j oint speed and the maximum TCP speed. The maximum joint speed can be set through **the motion parameter interface, while the maximum TCP speed is subjectto** s afety limitations (such as TCP speed, momentum, and stopping speed) . time and stopping distance.
- The robot 's maximum acceleration and jerk are affected not only by the jogging parameters, but also by whether **the torque constraint** function is enabled, thereby ensuring the robot's safety and stability during operation.

10.6.2 Motion parameters (Magician) E6)

The teaching settings are robot **jogging/inching** , speed , and maximum acceleration. **Reproduction settings** are configured as robot **running project** or TCP_Motion parameters during **IP motion commands . It supports setting** the maximum velocity, maximum acceleration, and maximum jerk for each joint or in a Cartesian coordinate system. Making changes , click **"Save"** to save the modified values, click **"Cancel** " to cancel the changes, and click " **Restore Defaults"** to reset the parameters to factory default values.

10.7 Attitude settings

This page is used to move the robot to various factory-preset postures.

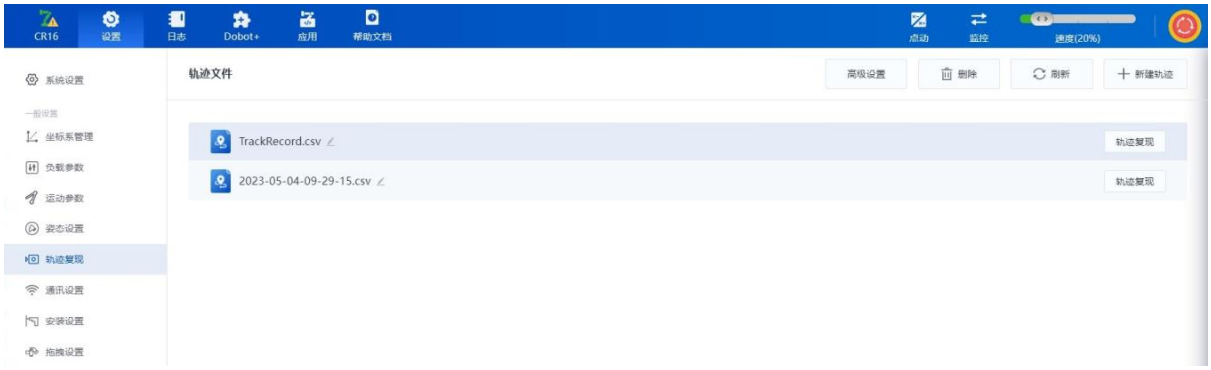
- **The packing posture** can reduce the space occupied by the robot and facilitate packing and transportation.
- **At zero-point orientation,** the angles of all joints are 0 degrees.

Long press **The movement** allows the robot to move to the corresponding posture.

> **illustrate :**
> The robot can also be moved to this posture through the point-and-click interface.

10.8 Trajectory Reproduction

Trajectory reproduction is used to record and reproduce the motion trajectory of a robotic arm.



Click the top right corner After **creating a new trajectory** , the robotic arm enters drag mode, allowing the user to drag the arm and record the dragged trajectory. One point is recorded every 50ms, with a maximum of 10,000 points recorded per trajectory. One point (approximately 500 seconds).

> **Notice :**
> Trajectory recording , otherwise the recorded trajectory will not be reproducible.

Desired trajectory is recorded , click " **Save** ." The robotic arm will exit drag-and-drop mode, and a new record will be added to the trajectory file list.

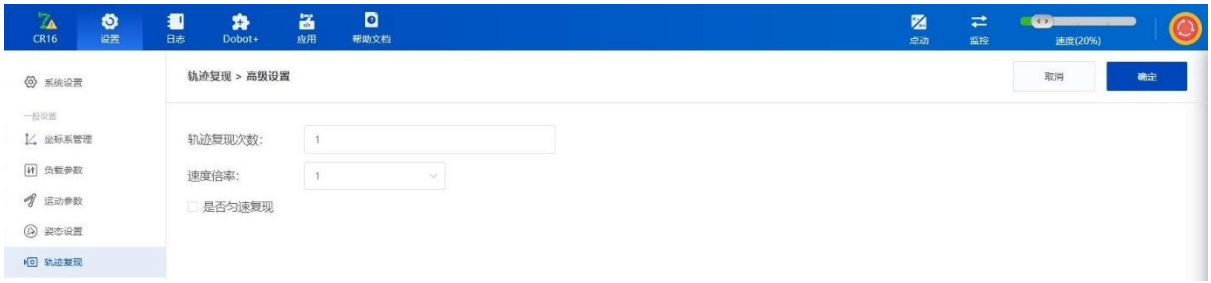
- Click next to the track file name to edit the track file name.
- Clicking **"Replay Track"** allows the robotic arm to reproduce the recorded track, and the r eproduction process can be stopped at any time.
- After selecting the trajectory file, click **Deleting** this track will remove it from the record.
- Click **Refreshing** the list of the latest trajectory files for the controller will retrieve the la test data.

> **illustrate :**
> Saved trajectory files can also be recalled using trajectory reproduction commands in graphical programming or script programming.

Clicking **"Advanced Settings"** allows you to configure the trajectory reproduction method. T he parameters set are accessible through CBSH Studio . The Pro interface and the trajectory of the button activation at the end of the robotic arm are effectively reproduced.

- **The trajectory reproduction count** indicates the number of times the robot will reproduce the trajectory after clicking **the trajectory reproduction button. When the count is greater than 1, the robotic arm** will automatically move its joints back to the trajectory starting point after completing one trajectory reproduction, and begin the next reproduction.

- **The speed multiplier** only takes effect **when the option to reproduce at a constant sp eed is not selected. The robotic arm reproduces** the trajectory after scaling proportionally at the original speed (unaffected by the global speed).
- After selecting **whether to reproduce at a constant speed** , the robotic arm will reproduce the trajectory at a constant global speed.



illustrate :

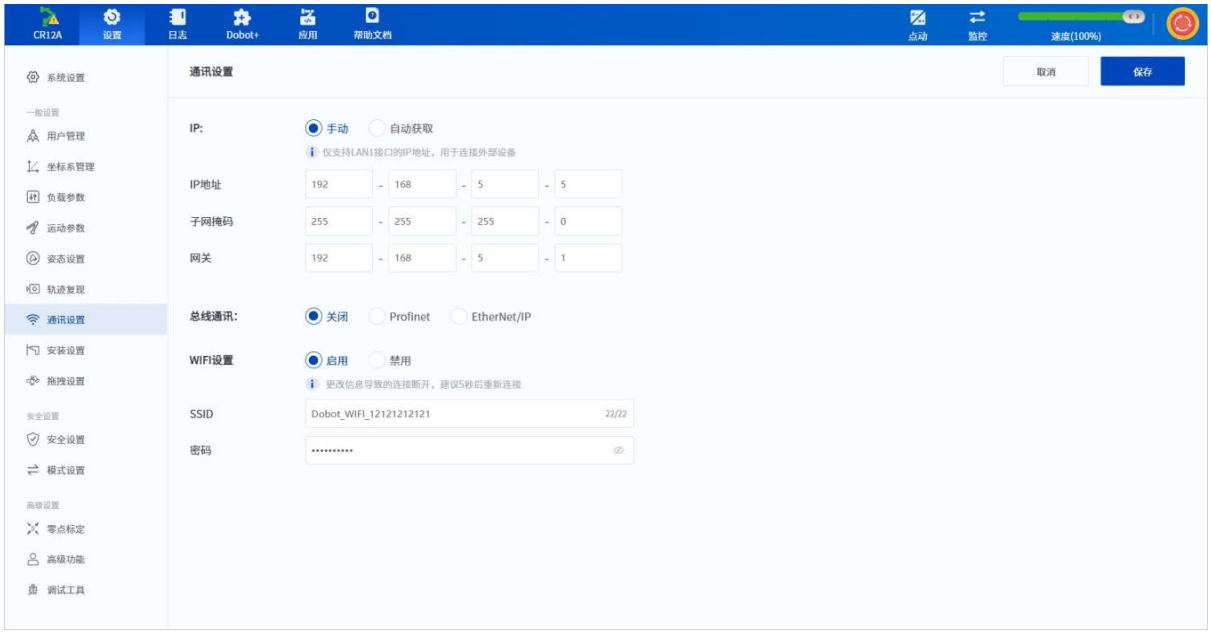
Users can also record trajectories via buttons on the robot's end effector (see the corresponding hardware manual for details). The differences between recording and reproducing trajectories via the CBSH Studio Pro interface and using buttons on the robot's end effector are as follows:

Via CBSH Studio The track file generated by the Pro interface is named after the save time (year-month-day-hour-minute-second), and a new file is created each time a track is saved . The track file generated by pressing buttons on the robot's end effector is named TrackRecord.csv, and each save overwrites the previous file.

via CBSH Studio The Pro interface allows you to select the trajectory to be reproduced, while the robot end effector can only reproduce the trajectory in TrackRecord.csv .

10.9 Communication settings

The communication settings are mainly used to configure the communication parameters of the currently connected controller, such as setting the IP address of the controller's LAN1, bus mode , and WiFi-related attributes.



IP settings

The robot can communicate with external devices via a LAN interface, supporting TCP, UDP, or Modbus protocols. Users can modify the robot. The IP address, subnet mask, and gateway of LAN1 port . When connecting external devices, the IP address must be on the same network segment as the external device's IP address and must not conflict with it.

- If the robot is connected directly to external devices or through a switch, select **manual mode** and modify the IP address, subnet mask (required when connecting multiple network segments), and default gateway to ensure the robot and external devices are on the same network segment.
- If the robot connects to external devices via a router, select " **Obtain an IP address auto matically** " (the router will automatically assign an IP address).

Bus communication

Configure bus communication functionality; supports setting to **disable** , **Profinet** , or **EtherNet/IP** .

Setting it to **Profinet** or **EtherNet /IP** , you need to configure the robot's actions when bus c ommunication is lost:

- **Continue execution** : If bus communication is lost, no action is taken and the project continues to run.
- **Pause** : Pauses the running project when bus communication is lost.

- **Stop** : Stops the running project when bus communication is lost.

For instructions on using the bus communication function, please refer to the " CBSH Bus Communication Protocol Document (EtherNet/IP, Profinet)".

⚠ **Notice :**
The bus communication to Profinet, the robot's LAN1 port can only be used for Profinet communication and its IP address cannot be configured. If you need to use LAN1 for other communication purposes, please modify and save the bus communication settings and then restart the control cabinet.

Wi-Fi settings

The robot can communicate with external devices via WiFi. Users can enable or disable WiFi, and modify the WiFi name and password. Click to view the current password.

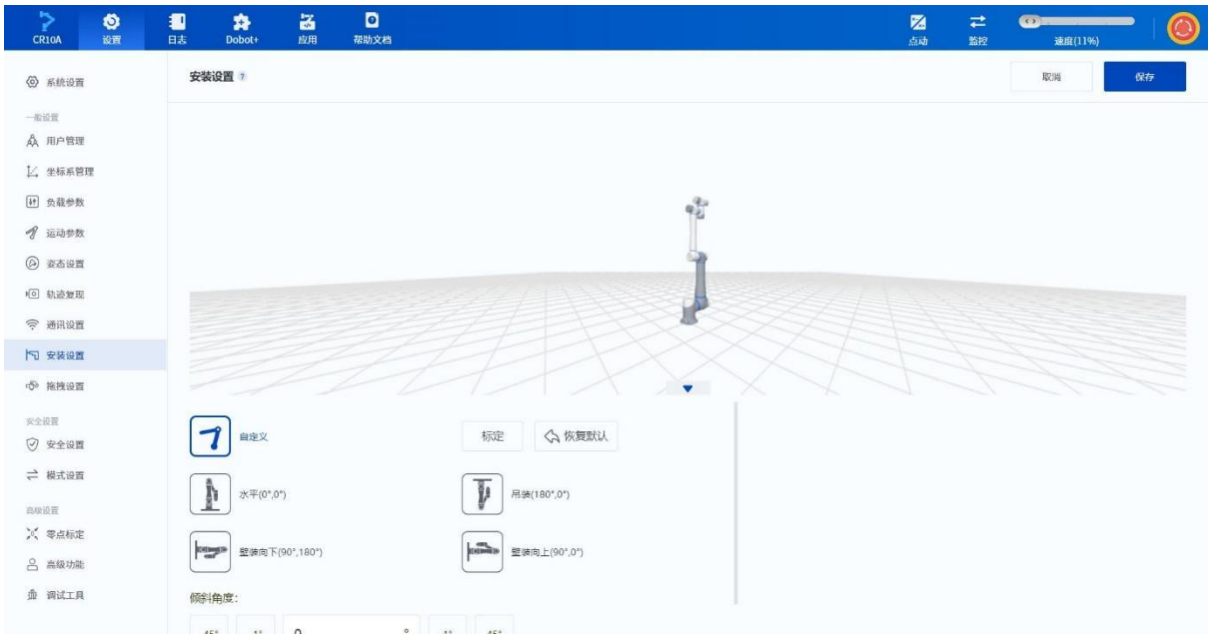
Magician The WiFi module for the E6 needs to be purchased and installed by the user and cannot be toggled on this page.

⚠ **Notice :**
Changing Wi -Fi settings may cause the software to disconnect from the robot. Pleas e try to reconnect after 5 seconds.

10.10 Installation settings

Normally , the robotic arm is installed on a stable table or ground, in which case no operation is required on this page. If the robotic arm is ceiling-mounted, wall-mounted, or installed at an angle, the rotation angle and tilt angle need to be set in the enabled state.
The installation settings is to inform the robot of the correct direction of gravity and to enable t he 3D model in the software to be displayed at the correct angle.

⚠ **Notice :**
Installation settings may result in collision detection errors or the device becoming uncontrollable when dragged.



Users can perform calibration either manually or automatically.

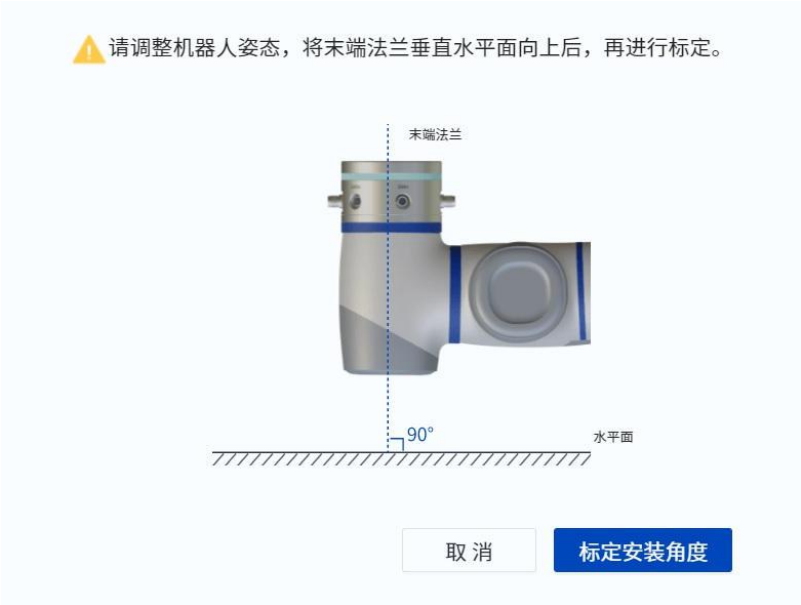
Manual calibration

Depending on the actual installation posture, select the corresponding installation posture on the left side of the page, or select **"Custom"** and then adjust the tilt and rotation angles below. Detailed explanations of each posture can be found by clicking the link to the right of the page title ? .

- **The tilt angle** refers to the angle at which the body rotates counterclockwise around the X-axis from the origin position.
- **The rotation angle** refers to the angle by which the body rotates counterclockwise around the Z-axis from the origin position.

Automatic calibration

After the robotic arm is installed and enabled, click **Calibrate** and follow the instructions in t he pop-up dialog box to obtain the tilt angle and rotation angle.



Clicking **"Restore Defaults"** will restore the calibrated angles to their default values.

setting the installation angle, please try entering drag mode via the button at the end o f the robotic arm to verify that the drag function is working properly. If there is any problem , please reset the installation angle or contact technical support.

10.11 Drag and drop settings

The drag settings are primarily used to adjust the sensitivity of each joint of the robot during dragging operations. Lower sensitivity results in greater resistance during dragging . The value range is 1% to 90%.

illustrate :

- The sensitivity setting only takes effect when the joint dragging speed has not reached the speed limit. Once the dragging speed reaches the speed limit, a reverse resistance will be generated to prevent overspeeding.
- Different robotic arm models. The speed limit values for J1~J3 are approximately 30~40°/s, while the speed limit values for J4~J6 are approximately 90 ~100°/s.

10.12 Power supply voltage (DC control cabinet/Magician) E6)

When the robot is connected to a DC control cabinet or Magician In E6, you need to use CBSH Studio . Setting power in Pro software Voltage. This voltage range is related to the power supply function (used to release the electromotive force generated when the robot decelerates or brakes). Please set it according to the actual voltage range of the input power supply to avoid overvoltage protection shutdown or damage to the control cabinet.

Click **Settings** > **Power supply voltage** , the range of the actual input power supply voltage.

- Connect CC262 For the DC version, the value range is 30~60V, and the minimum value is less than or equal to the maximum value.
- Connect to Magician When E6 is used, the value range is 36~58V, and the minimum valu e is less than or equal to the maximum value.

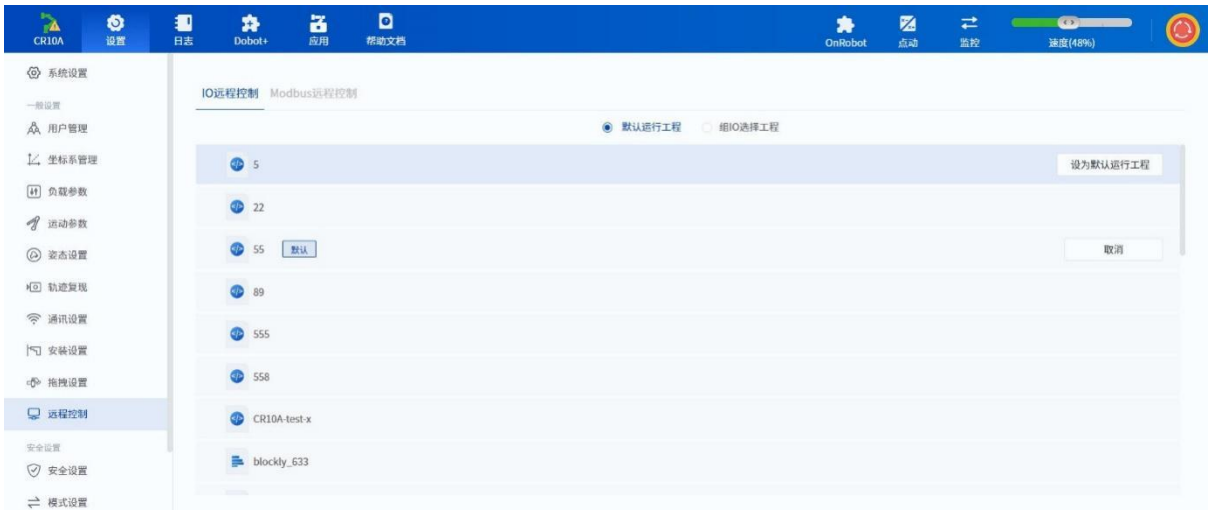
10.13 Remote control

This page is used to configure the project for remote control IO monitoring and Modbus monitoring.

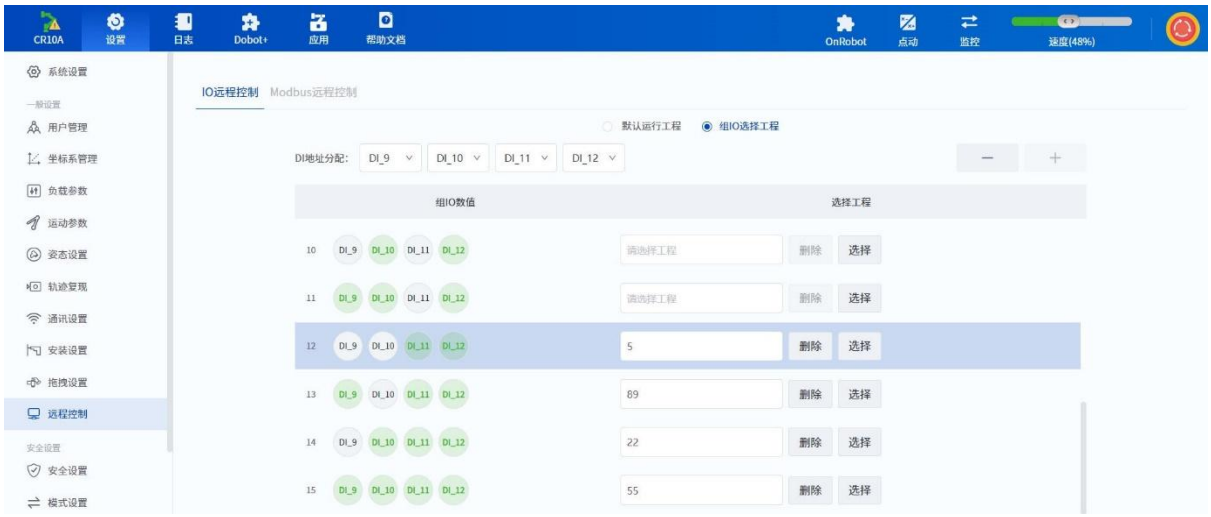
10.13.1 IO remote control

Default running project

You select a project from the list and click **"Set as Default Run Project" on the right** , the function is configured to run the selected project directly when **the initial** DI is triggered .



Clicking **the Cancel** button will cancel the currently selected project. **Group IO Project Selection** by choosing **a group IO** , multiple projects can be configured using the group IO method.



d to the group I/O. The more addresses assigned (up to 4), the more projects can b e configured.

- One address: can be configured for two projects.
- 2 addresses: 4 projects can be configured
- 3 addresses: configurable for 8 projects
- 4 addresses: configurable for 16 projects

2. Through **the DI address assignment** drop-down box. The address assigned to group I/O cannot be the same as that of remote I/O or secure I /O (CCBOX).

3. Assigning addresses, you can configure the project for each group of I/O. You can set it up according to your actual needs, or leave it blank.

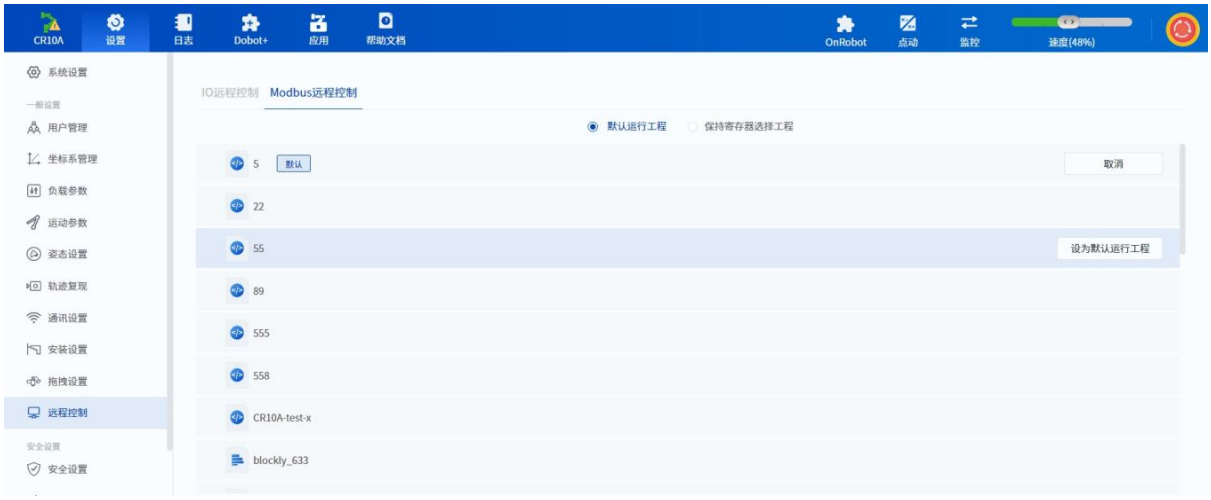
   Running a project , select the corresponding project by setting the value of the corresponding group IO (green indicates ON, gray indicates OFF). If no project is selected for the corresponding group IO, the robot will not run the project and the controller will generate an alarm when the DI is **triggered** .

> **Case Study :**
> Taking the allocation of four addresses DI9~DI12 in the above diagram as an example:
> - DI9 and DI10 being OFF, and DI11 and DI12 being ON, indicate that project 5 is selected;
> - If DI10 is OFF and DI9, DI11, and DI12 are ON, then project 89 is selected.
> - If DI9 is OFF and DI10, DI11, and DI12 are ON, then project 22 is selected.
> - If all DI9 to DI12 are ON, then project 55 is selected.

10.13.2 Modbus remote control

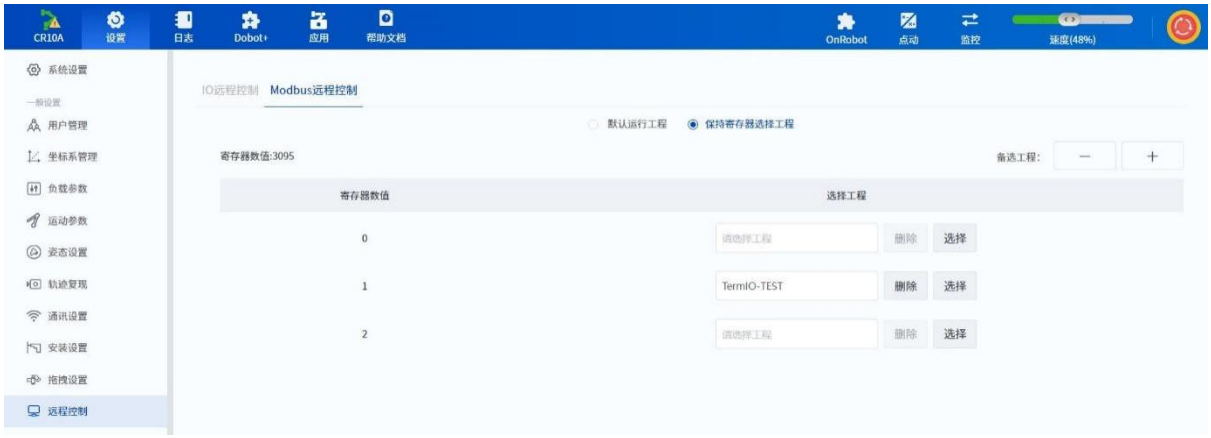
Default running project

You select a project from the list and click **"Set as Default Run Project" on the right** , the f unction is configured to run the selected project directly when the **starting** coil register is t riggered .



Clicking **the Cancel** button will cancel the currently selected project. **(Keep register selected project)**

Switch to **the holding register selection project** page to configure multiple projects.



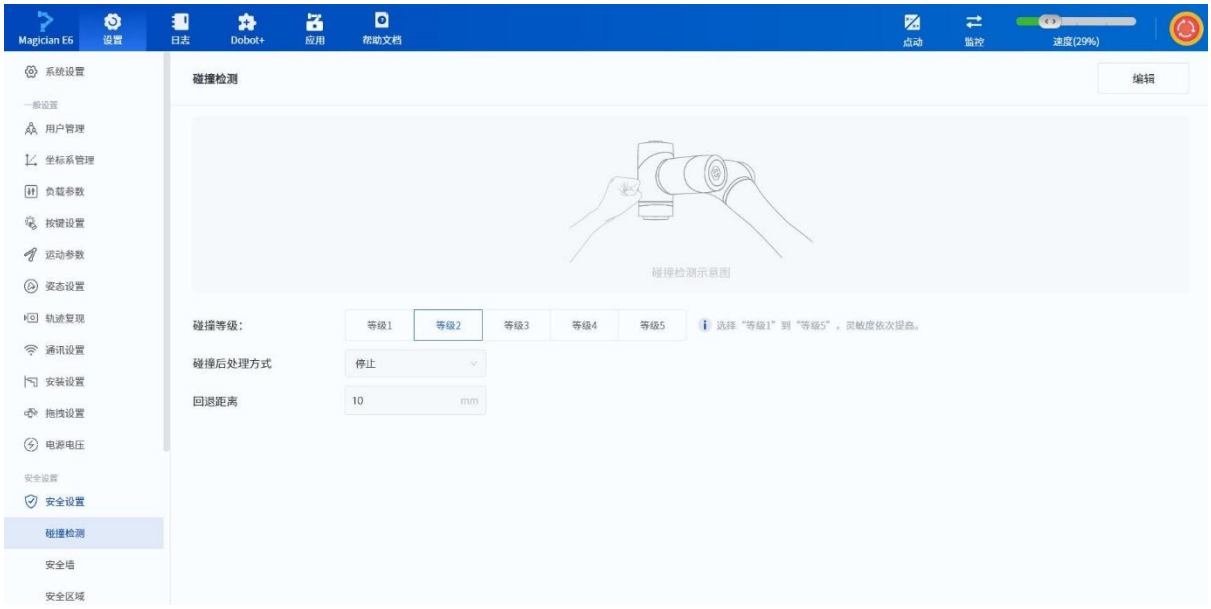
Click **+** or **-** The button can increase or decrease the number of projects to be configured (up to 256). When **the start** coil register is triggered, the project to be started is determined based on the value of the holding register at the specified address (3095).

10.14 Security Settings

10.14.1 Collision detection (Magician) E6)

The robot will automatically stop if a collision is detected during its movement. Users can set the sensitivity of the collision detection and the specific handling method after a collision.

> **illustrate :**
> If you need to enable or disable collision detection, please contact technical support.



According to actual needs; the higher the collision level, the less force is required to trigger collision detection.

- Level 5 is only applicable to scenarios where the robot operates at low speeds. At thi s collision level, the robot may falsely trigger collision detection when it is running at high speed and high acceleration under load.
- When the robot is fully loaded, please set the collision level to level 3 or lower.The way robots handle collisions differs between inching and automatic operation.

Collision occurs during inching

The software pop-up window indicates a collision has been detected. You need to resolv ethe cause of the collision and click **"Reset"** . If resolving the collision requires operating the control software , you can click **"Remind me in one minute" to** temporarily close the p op-up window (the pop-up window will reappear after one minute).

Collision occurs during automatic operation

The robot handles the situation according to **the post-collision processing method :**

- **Stop** : The robot has stopped operating.
- **Pause** : The robot is paused. Depending on the situation, you can choose to resolve the collision cause and continue, or stop. While the robot is in a collision pause state, you can resume operation by briefly pressing the drag-and-teach button on the robot's e nd effector.

Regardless of the method of handling, after a collision, the robot will automatically retreat a specified distance based on the trajectory before the collision. The retreat distance can be set from 0 to 50 mm, with a default value of 10 mm.

10.14.2 Security Limitations (CRA Series)

To ensure the safety of both the robot and the user, the robot system will limit relevant parameters during robot movement, and users can set the limit values according to their own needs.

Parameter settings

It supports both quick settings and custom settings.

- **Quick settings** : Select from 5 preset levels (hereinafter referred to as Level 1, the most lenient level, Level 5, and so on).

> **illustrate :**
> - Level 5 is only applicable to scenarios where the robot operates at low speeds. At this collision level, the robot may falsely trigger collision detection when it is running at high speed/high acceleration under load .
> - When the robot is fully loaded, please set the collision level to Level 3 or lower. The preset level is only a suggestion and does not replace a proper risk assessment.

- **Custom** : Enter the values for each restriction parameter yourself. Switching back to quick settings will reset the parameters to the default values for the corresponding level. The meanings of each restriction parameter are as follows (all parameters need to be set separately for normal mode and reduced mode).
- **TCP force** : The maximum force that can be applied to the center point (TCP) of the robot's end effector.
- **Power** : Limits the maximum total power of the robot.
- **TCP Speed** : Limits the maximum TCP speed at which the robot can run.
- **Momentum** : Limits the maximum overall momentum of the robot during operation.
- **Stop time** : The maximum time required for the robot to stop after an alarm is triggered (including emergency stop).
- **Stopping distance** : The maximum distance required to stop the robot after an alarm is triggered (including emergency stop).

the robot adjusts the planned maximum speed based on the actual situation to ensure that the parameters mentioned above remain within the limits. Under this premise, if **the TCP force** or **power exceeds the limit** during robot movement , it will be considered a collision, and the robot will automatically stop and trigger a collision detection alarm.

illustrate :

TCP speed , momentum, stopping time, and stopping distance together limit the robot's maximum TCP speed. The larger the parameter, the higher the robot's movement speed.

The way robots handle collisions differs between inching and automatic operation.

Collision occurs during inching

The software pop-up window indicates a collision has been detected. You need to resolve the cause of the collision and click **"Reset"** . If resolving the collision requires operating the control software , you can click **"Remind me in one minute" to** temporarily close the pop-up window (the pop-up window will reappear after one minute).

Collision occurs during automatic operation

The robot handles the situation according to **the post-collision processing method :**

- **Stop** : The robot has stopped operating.
- **Pause** : The robot is paused. You need to choose whether to continue after resolving the collision cause, or stop, depending on the situation. While the robot is in a collision -paused state, you can also resume operation by briefly pressing the drag-and-teach button on the robot's end effector.

Regardless of the method of handling, after a collision, the robot will automatically retreat a specified distance based on the trajectory before the collision. The retreat distance can be set from 0 to 50 mm, with a default value of 10 mm.

Click **"Edit"** to modify the parameters. After modification, click **"Save** " to save the modified values. Click **"Cancel** " to cancel the current modification. Click **"Restore Defaults "** to reset the parameters to factory default values.

> **illustrate :**
> The value in reduced mode must be smaller than that in normal mode, otherwise it cannot be saved.

10.14.3 Joint limiting (CRA series)

Users can set soft limits for each joint of the robot to restrict the range of motion of each joint.

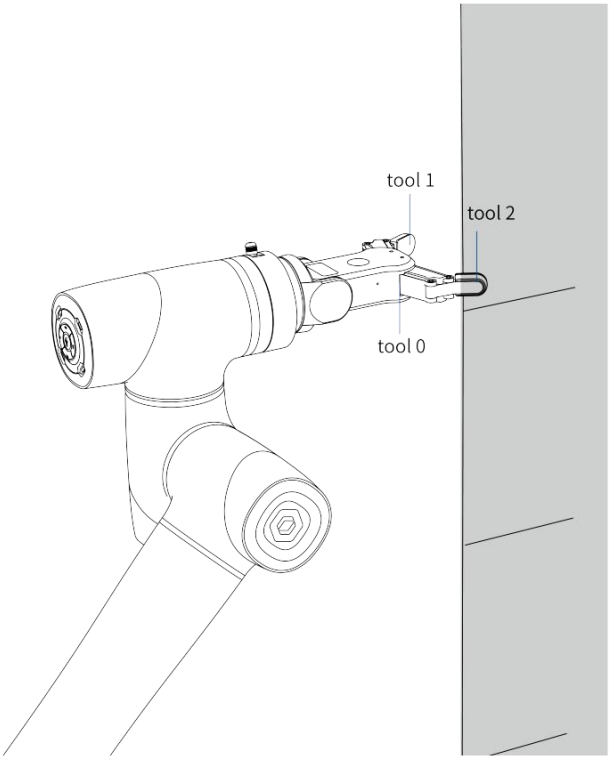
When the robot's actual joint angle is less than (negative limit + negative limit tolerance) or greater than (positive limit + positive limit tolerance), an alarm will be triggered and the robot will stop moving.

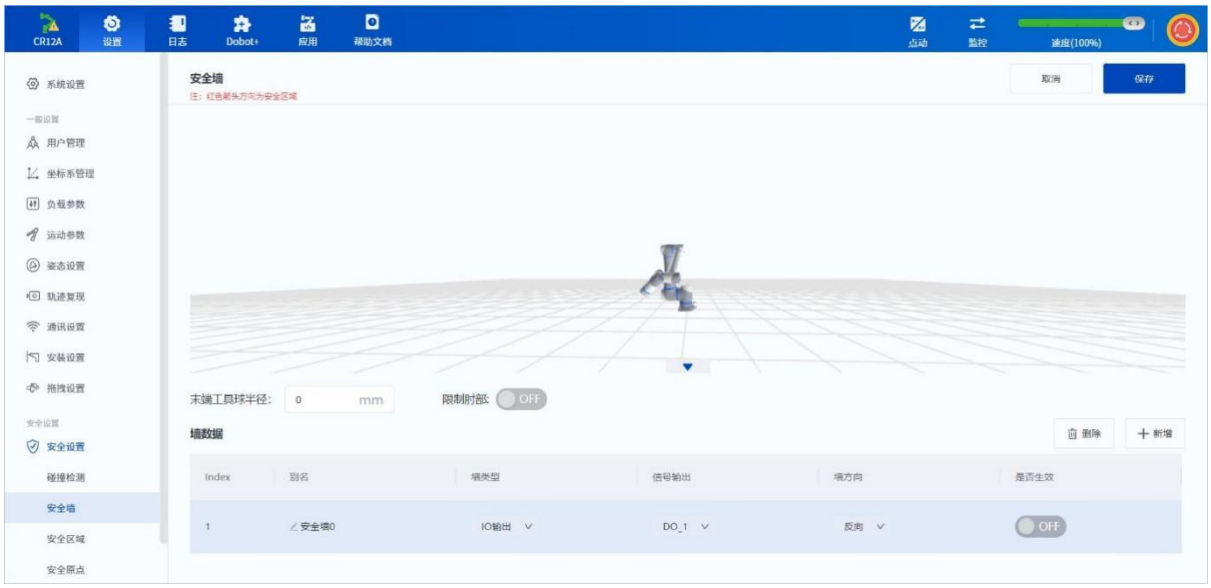
Negative and positive limits can be modified.

Click **"Edit"** to modify the parameters. After modification, click **"Save** " to save the modified values. Click **"Cancel** " to cancel the current modification. Click **"Restore Defaults "** to reset the parameters to factory default values.

10.14.4 Safety wall

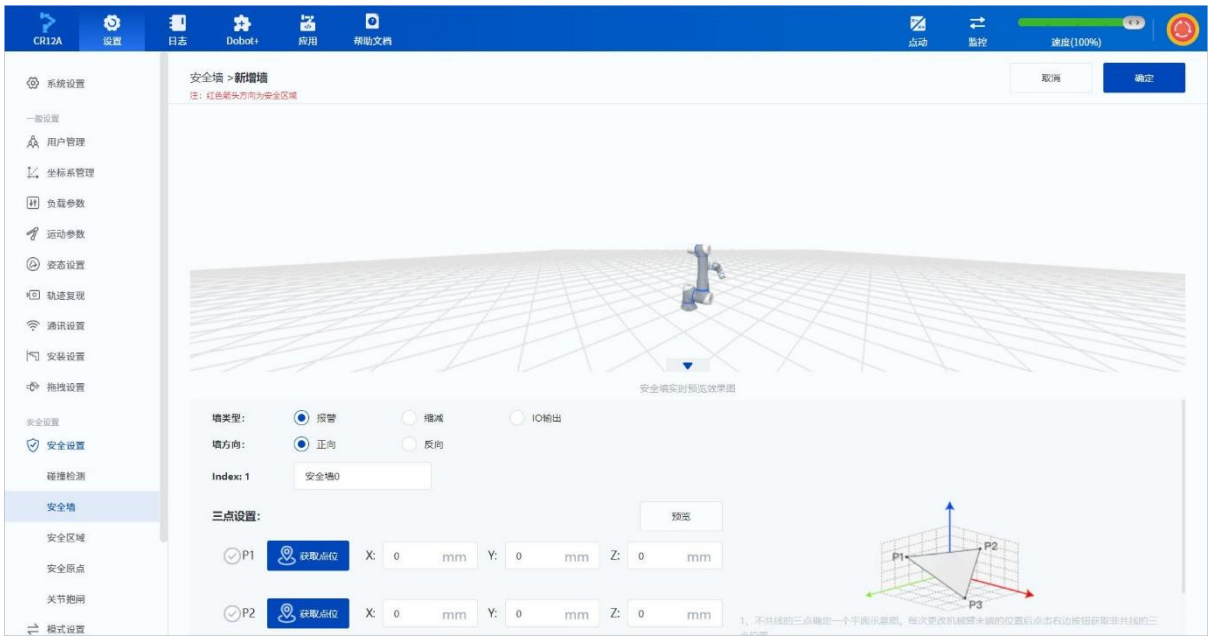
CBSH Studio Pro supports setting up to 8 safety walls, specifying the robot's safety space by the wall direction. When the robot's current tool coordinate system approaches or exceeds the safety space defined by a safety wall, different actions (such as alarms, deceleration, or triggering a specified DO) will be triggered depending on the set **wall type** . As shown in the figure below, if the current tool coordinate system is tool 0 or tool At 1 o'clock, it was determined that the robot was safe. Inside the wall; if the tool coordinate system is switched to tool If 2, then the robot is determined to have exceeded the safety wall.





Add security wall

Click ＋**Adding** a new safety wall is possible. A safety wall is a user-defined plane that can be determined by three non-collinear points on that plane . The user needs to dete rmine these three points first, hereinafter referred to as P1, P2, and P3.



1. Move or drag the robot to P1, then click. **Get the location** , get the coordinates of P1.

2. Use the same method to obtain the coordinates of P2 and P3, and then click **Preview** to see the generated safety wall in the simulation area above.

illustrate :

- Users can also manually enter or modify the Cartesian coordinates of the points.
- After modifying the settings, you need to click **Preview** for the simulation area display to update.

3. Sets the action to be triggered when the robot's current tool coordinate system approaches or exceeds the safe space defined by the safety wall.
   - **Alarm** : When approaching the safety boundary, if the current motion plan would cause the robot to exceed the safety range, an alarm will be triggered and the robot will stop moving.
   - **Reduction** : When the robot exceeds the safe space, it triggers the reduction mode and slows down its operation.
   - **IO Output** : When the robot exceeds the safe space, it triggers the designated DO (Directional Action) without affecting its movement. Different safety walls or safe zones can be active simultaneously, and the same DO can be set. Triggering any active safety wall or safe zone will trigger the corresponding action .
4. Set the wall orientation. You can view the wall orientation in the simulation area. The direction indicated by the arrow is the safe side of the wall, and the opposite direction is the restricted side.

illustrate :

The positive direction of the wall is based on P1. -> P2 The vector direction of ->P3 is the plane normal vector calculated according to the right-hand rule.

5. Click **OK** to add the security wall.

Modify the security wall

All safety wall properties except Index **can be modified. Signal output is configurable** only for walls with **an IO output type** . The switch on the right controls whether the safety wall is active (only operable in the enabled state under robot operation). Only active safety walls will interfere with the robotic arm and be displayed in the simulation area.

Select the security wall and then click The selected security wall can be deleted .

Advanced settings

End tool ball radius

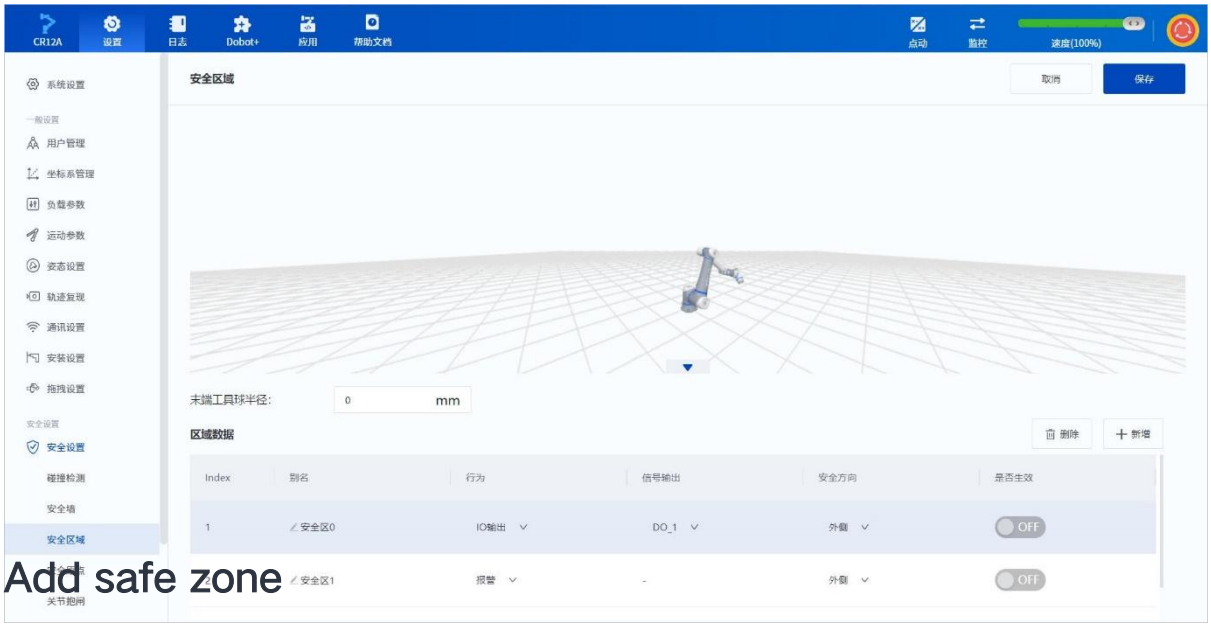
Specifies the radius of the end tool sphere (a spherical space with a custom radius centered on TCP). When this spherical space interferes with the safety wall restriction area, it will trigger the safety wall action; setting it to 0 means that only TCP interferes with the restriction area.

Limit elbow

When this option is set to **ON** This indicates that the J3 joint will also interfere with the restricted area, and is set to... **OFF** Only the end-effector ball will interfere with the restricted area.

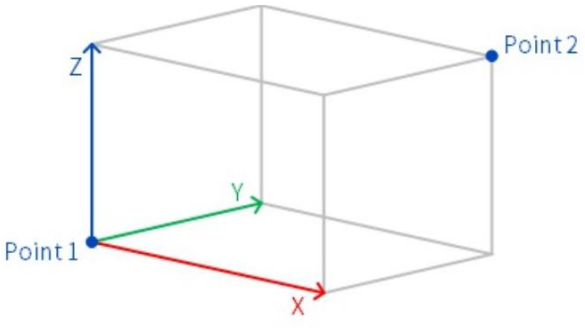
10.14.5 safe zone

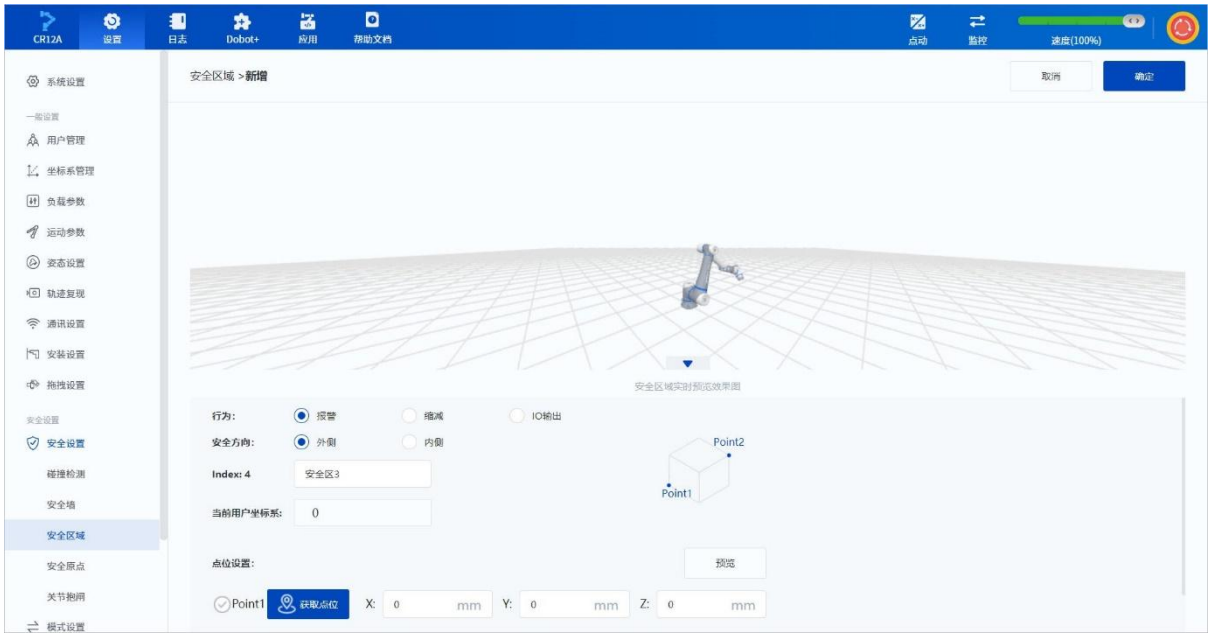
CBSH Studio Pro supports setting up to 6 safe zones, with the inner or outer edge of a designated zone defined as the robot's safe zone. When the robot's current tool coordinate system approaches or exceeds a safe zone, different actions will be triggered based on the set **behavior** (such as alarm, deceleration, or triggering a specified DO).



Add safe zone

Click ＋**Add** a new safe region. The safe region is a cube. The user needs to teach the two opposite vertices of the safe region (Point1 and Point2), and then determine the orientation of the cube's edges by specifying the user coordinate system, as shown in the figure below.





1. Move or drag the robot to Point1, then click. **Get the point location** , get the coordinates of Point1.

2. Use the same method to obtain the coordinates of Point2.

3. Select **the current user coordinate system** , and then click **Preview** to see the generated safe area in the simulation area above.

illustrate :

- Users can also manually enter or modify the Cartesian coordinates of the points.
- After modifying the settings, you need to click **Preview** for the simulation area display to update.

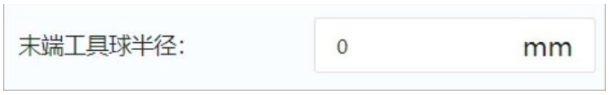
4. Sets the action to be triggered when the robot's current tool coordinate system approaches or exceeds the safe zone.
   - **Alarm** : When approaching the safety boundary, if the current motion plan would c ause the robot to exceed the safety range, an alarm will be triggered and the robot will stop moving.
   - **Reduction** : When the robot exceeds the safe space, it triggers the reduction mode a nd slows down.
   - **IO Output** : When the robot exceeds the safe space, it triggers the designated DO (Directional Action) without affecting its movement. Different safe zones or safety walls can be active simultaneously , and the same DO can be set. Triggering any active safe zone or safety wall will trigger the corresponding action .
5. Set the safe direction. **The outer side** indicates that the area outside the cube is a safe area, and **the inner side** indicates that the inside of the cube is a safe area.
6. Click **OK** to add a safe zone.

Modify safe zone

All safe zone attributes except **Index** and **region shape can be modified, among which** the safe zone for **signal output** with only **IO output behavior can be configured.** The switch on the right controls whether the safety zone is active (this can only be operated in the enabled state under robot conditions). Only active safety zones will interfere with the robot and be displayed in the simulation area.

After selecting the safe area, click **Delete** the selected safe area.

Advanced settings

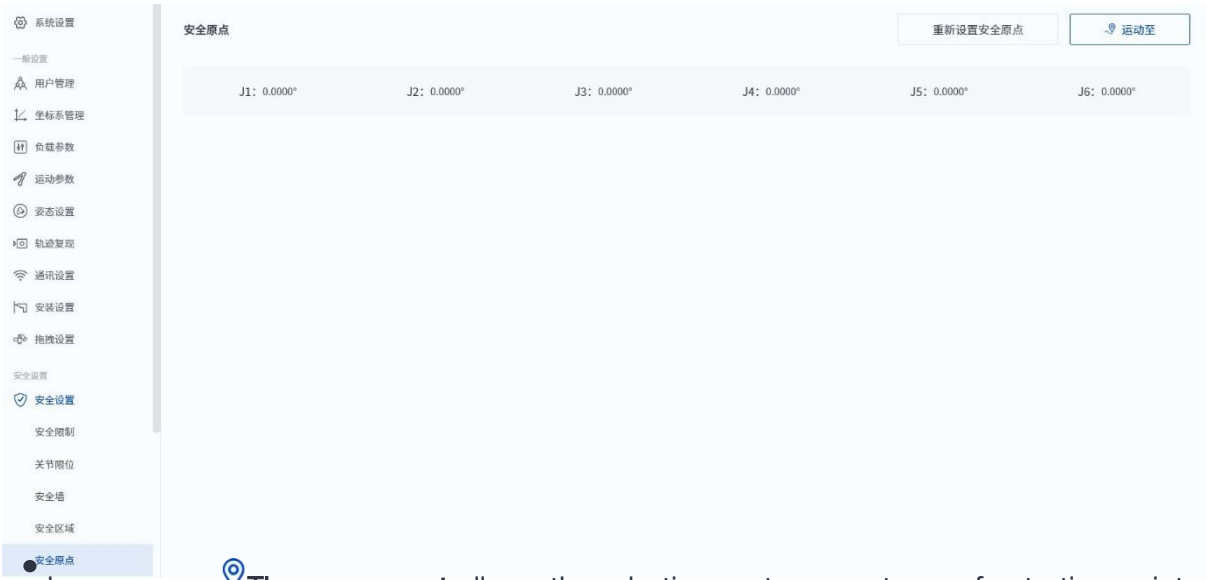


End tool ball radius

Specifies the radius of the end tool sphere (a spherical space with a custom radius centered on TCP). When this spherical space interferes with an unsafe region , it will trigger the action of the safe region; setting it to 0 means that only TCP interferes with the restricted region.

10.14.6 Safety origin

The safety origin is a customizable pose, with a default pose of zero, meaning all joint angles are 0. Users are advised to modify this pose according to their application . When the robot is at the safety origin, the safety origin status can be output via safety I/O , system I/O , or Modbus. The system uses state signals, which users can use to make judgments and implement logic such as allowing the robot to run only when it is at a safe origin.



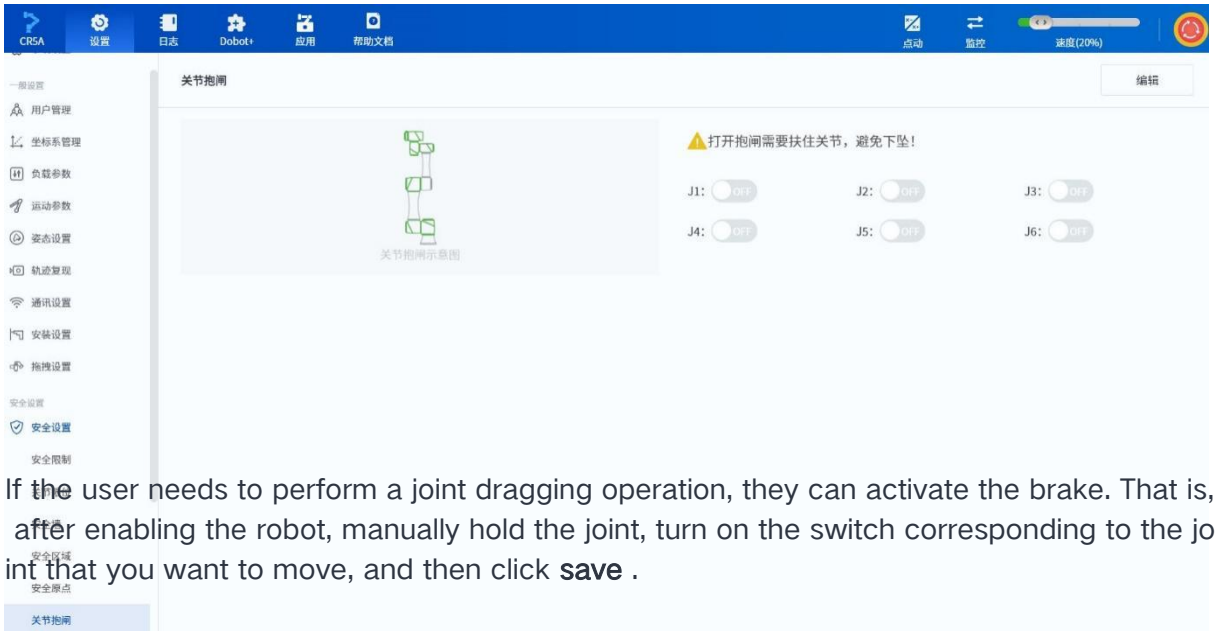
- Long press **The movement** allows the robotic arm to move to a safe starting point.
- Click **"Reset Security Origin"** to modify the security origin.

You can manually input the angles of each joint, or control the robot to move to a specified posture and then click. The current angles of the person 's joints. Clicking " **Restore Default Points"** will restore the safe origin to the default position.

Determining the angles of each joint, click **Save** to update the safety origin.

10.14.7 Joint brake

The robot is in the disabled state, to prevent joint movement, the joints will automatically engage brakes to keep the motors locked in position, ensuring that the moving parts of t he machine will not move due to their own weight or external forces. In an emergency, the user can release the joint brakes through this page and then drag the corresponding joint to move it.



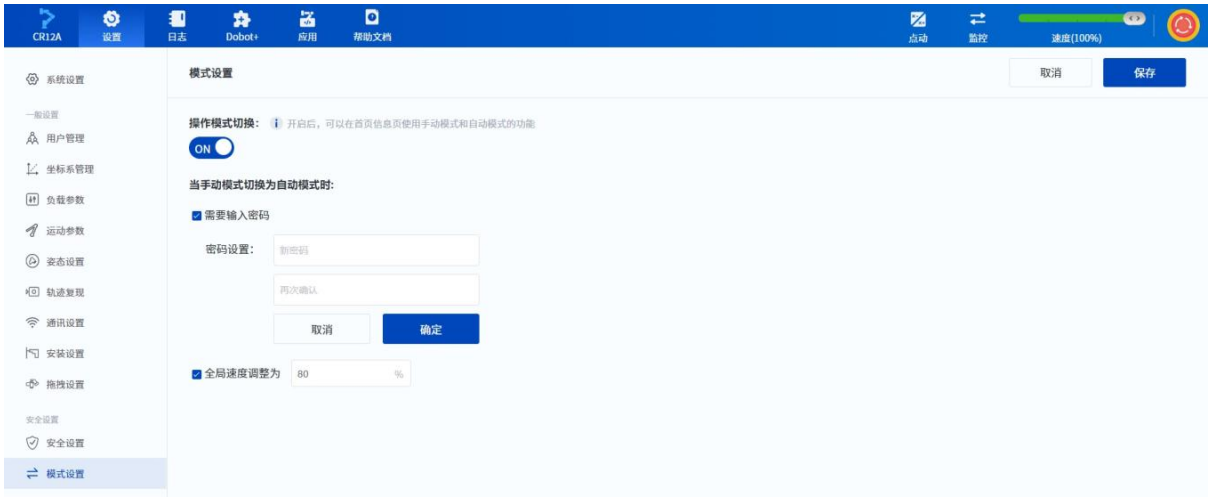
If the user needs to perform a joint dragging operation, they can activate the brake. That is, after enabling the robot, manually hold the joint, turn on the switch corresponding to the jo int that you want to move, and then click **save** .

warn :

When the brake is engaged , the joint must be manually held to prevent the robotic arm f rom slamming down due to its own weight.

10.15 Mode settings

The robot's manual /automatic mode switching function is disabled by default at the factory (CR20 is in manual mode by default), and can be enabled on this page. Manual/ automatic mode is primarily used to improve safety in field applications; please select whether to enable it based on the risk assessment results.



Enabling **the operation mode switching function, you can choose whether a password is r equired** when switching from manual mode to automatic mode . If this option is selected , a password needs to be set (no initial password).

If **the global speed adjustment** function is selected, the global speed will automatically adj ust to the set value when switching to automatic mode. **Global speed adjustment** can onl y accept integer values from 1 to 100.

10.16 Zero point calibration

When the transmission components of the robotic arm, such as the motor and reducer, are replaced, or when it collides with the workpiece, the zero point position of the robotic arm changes , and the zero point calibration of the robotic arm is required.

Zero point calibration procedure:

1. According to the on-screen instructions, first move the robotic arm to the zero-point position (you can use the zero-point stickers on each joint of the robotic arm to confirm the position; see the corresponding robotic arm's hardware user manual for details).
2. With the robot enabled, click the **"Calibrate All Joints"** button in the upper right corner of the page to calibrate all joints, or click the **"Zero Point Calibration " button corresponding** to a single joint to calibrate that single joint.
3. A pop-up window prompts you to enter the manufacturer's password.

> **illustrate :**
> If you remain on the current page to continue the calibration operation, you do not need to re-enter the manufacturer's password; however, if you switch to another settings page and then return to this page to perform calibration, you will need to re-enter the manufacturer's password.

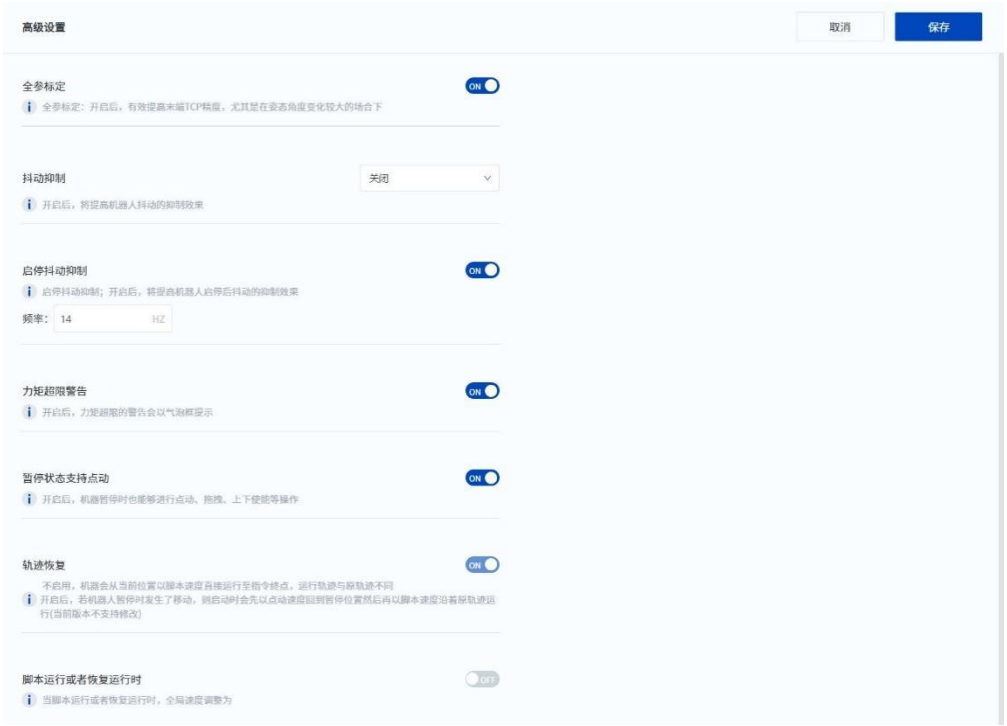
4. the manufacturer's password , click **"OK" and** a second pop-up window will prompt you about potential problems after calibration. After confirming the operation, click **" Zero Point Calibration** ".

> **Danger :**
> Zero-point calibration is only used when the zero-point position changes; please opera te with caution.

5. successful calibration , the robotic arm will automatically be enabled. You can then view the joint coordinates on the jog panel; at this point, the values of J1 to J6 will all be 0 .

10.17 Advanced features

When advanced functions need to be toggled due to site conditions, they can be configured on this interface. Please refer to the on-screen instructions for descriptions of each function ; unless there are specific requirements, it is recommended to keep the default values.



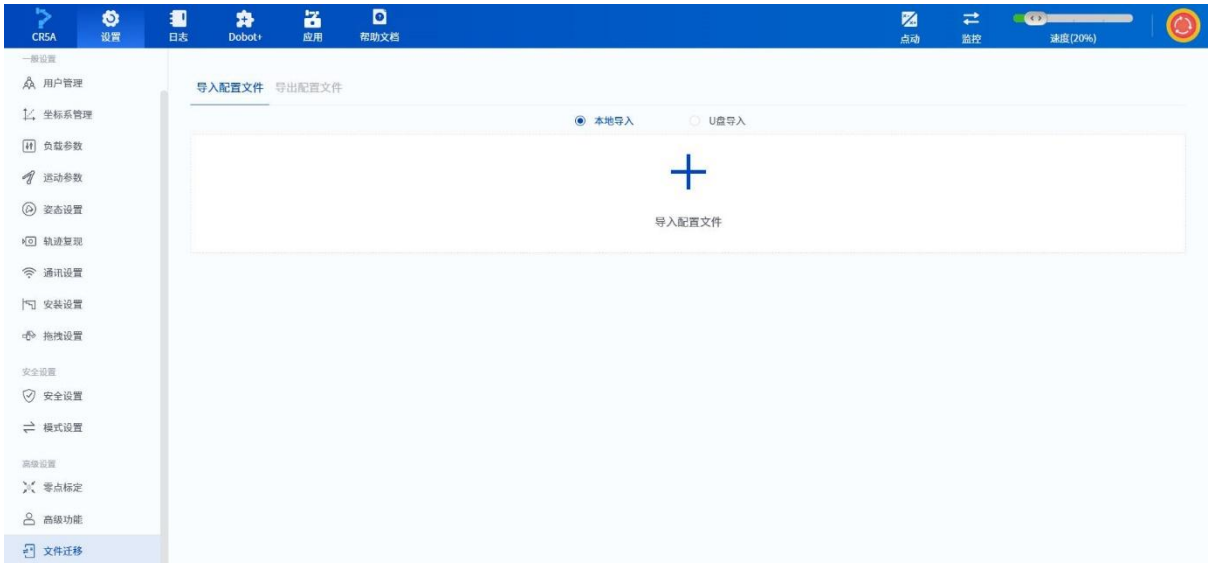
illustrate :

- **The paused state support for jogging** and **trajectory recovery** functions, please see **the Track Recovery section** .
- When connecting a CR20A/CR20V robot, a switch for **a limiting overload operation mode** will appear . Only administrators can enable this function. When enabled, a warning window will pop up and check if the torque constraint switch is currently enabled. If not activated, the torque constraint switch will be forcibly activated.

10.18 File migration

CBSH Studio Pro supports importing and exporting internal robot files (such as program files and configuration files). It can be used for scenarios such as system backup and device duplication, and supports importing and exporting via local storage o r USB flash drive.

Import configuration file



Click **Settings** > **The file migration page** leads **to the import configuration file** page, where you can import configuration files saved locally or from **a** USB **drive** into the system.

illustrate :

- Importing the configuration file, the robotic arm must be disabled before you can continue operating.
- Importing from a USB drive, only compressed files in the root directory of the USB drive can be displayed. Please select the correct file.

Export configuration file

Click the **"Export Configuration File"** tab, select the parameters and project files you want to export, and click the **"Export "** button. A pop-up **export** prompt window will appear.

Choose to export the system configuration file to **the control cabinet USB drive** or **local drive** .

illustrate :

- If you choose to export to **the control cabinet USB drive** , the default export location is the root directory of the USB drive.
- When using the control cabinet's USB drive for import/export, the user can insert the USB drive into any USB port on the control cabinet; if two USB drives are plugged into both ports, the system will default to reading the folder from the first USB drive found.
- Of exported projects is limited to 100.

10.19 Firmware upgrade

The **firmware upgrade** page allows you to upgrade the robot's firmware to the latest version with one click, or revert to the previous firmware version.

Click **Settings** > **Firmware upgrades** can be performed by importing the upgrade packag e file stored locally or on a USB drive **into** the **system** .

illustrate :

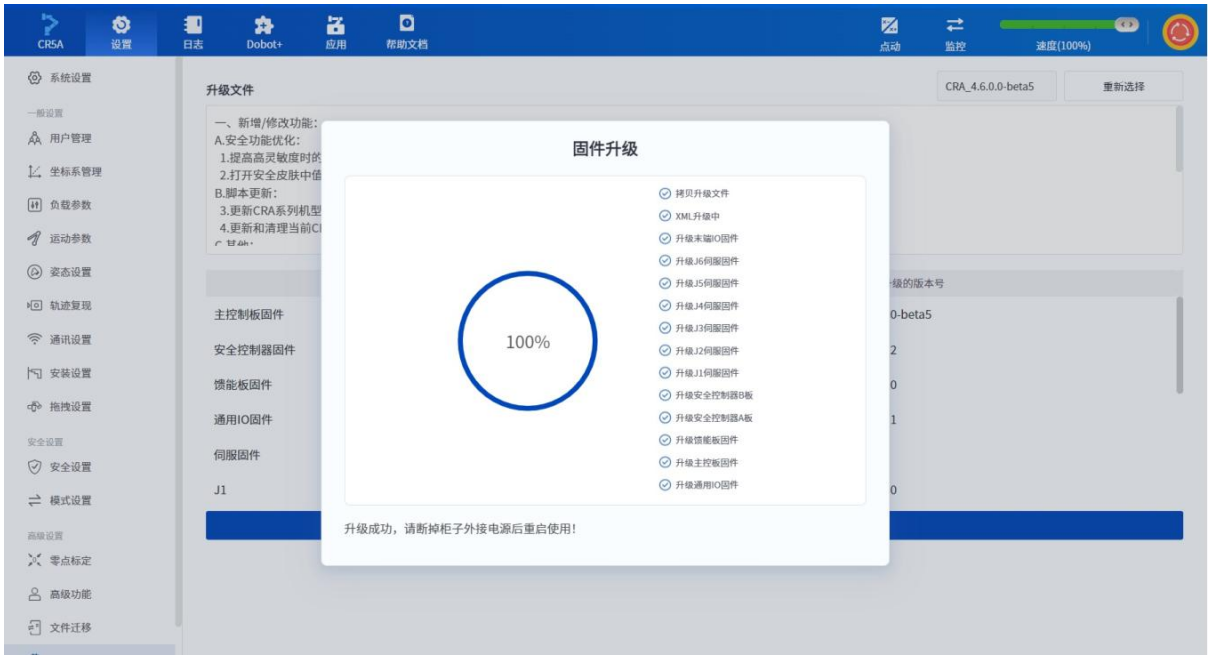
- Importing the upgrade package file, the robotic arm must be de-enabled before operation can continue; the robotic arm must not be powered off during the upgrade process .
- Using a **USB flash drive** , it is recommended to plug the USB flash drive into a USB docking station without a network interface. Only compressed files in the root directory of the USB flash drive are supported during the import process
- Please select the correct file.

- **Firmware upgrades** can only be performed by administrators , and this permission cannot be configured.

After importing the upgrade package file, extract the file to your local machine. Once the extraction is successful, the old and new firmware version information and upgrade details will be displayed.

Click **"One -Click Upgrade"** to start the version upgrade. The interface will display the upgrade progress. At this time, other functions will be unavailable.

- If you receive a message saying "Upgrade file copy failed" during the upgrade process, you can click " **Upgrade Again"** to try upgrading again with one click.
- If any errors other than "Upgrade file copy failed" are displayed during the upgrade process, resolve them according to the error information returned by the interface.
- After the upgrade is complete , a message will indicate that the upgrade was successful. You will need to press the rocker switch to power off and restart the control cabinet. After restarting the control cabinet, connect the robot to CBSH Studio. Pro reconnects.



illustrate :

Prioritize using robot maintenance tools for firmware version upgrades.

Appendix A Modbus register definition

1 Introduction to Modbus

Modbus is a serial communication protocol that allows robots to communicate with external devices. When controlling a robot via an external device such as a PLC , the external device acts as the Modbus master, and the robot acts as the Modbus slave.

Several Modbus solutions :

- Modbus -RTU: Compact, using binary representation of data. It employs a checksum me thod based on cyclic redundancy check.
- Modbus -ASCII: A string representation based on ASCII codes, which is a readable and redundant representation. It uses a checksum method based on vertical redund ancy check.
- Modbus-TCP: Connects via TCP/IP using TCP. This method does not require checksum ca lculation.
- Modbus-RTU-over-TCP: Data packets containing RTU

content transmitted using TCP. We currently offer Modbus-TCP

and Modbus-RTU-over-TCP solutions. According to the

Modbus protocol, the Modbus address assigned to the CBSH six

-axis collaborative robot is as follows:

- 00001-09999: Coil registers, which can only take values of 0 or 1, are used to control the robot, and can be read and written.
- 10001-19999: Contact registers, values can only be 0 or 1, used to obtain robot status, re ad-only.
- 30001-3 9999: Input register, supports a maximum value of 64-bit double-precision floating-point number, used to obtain real-time feedback data from the robot , read -only.
- 40001-49999 : Holding registers, supporting a maximum value of 64-bit double-pr ecision floating-point numbers, used for robot PLC interaction, readable and

writable. various registers follow the standard Modbus protocol:

寄存器类型	读取寄存器	写单个寄存器	写多个寄存器
线圈寄存器	01	05	0F
触点寄存器	02	-	-
输入寄存器	04	-	-
保持寄存器	03	06	10

The robot provides two tables for external devices to access; map1 can be accessed via 502 (Modbus-TCP) and 503 (RTU-over-... Map2 can be accessed via TCP ports , while map2 can be accessed via ports 1502 (Modbus-TCP) and 1503 (RTU-over-TCP). Its resource description is as follows:

Parameters/Value	5 02	5 03	1 502	1 503
Number of coils	1 0000	1 0000	1 0000	1 0000
Number of discrete inputs	1 0000	1 0000	1 0000	1 0000
Number of input registers	1 0000	1 0000	1 0000	1 0000
Number of Registers	1 0000	1 0000	1 0000	1 0000
The map	m ap1	m ap1	m ap2	m ap2
Bind port	5 02	5 03	1 502	1 503
Modbus protocol	TCP	RTU - over-TCP	TCP	RTU - over-TCP
Maximum number of main sites supported	20	20	20	20
From station ID	1	1	1	1

Of these, some addresses in map1 are used by default by the robot system; see the register definitions below for details. map2 is currently empty and can be used by the user as needed.

2 Coil register (map1, for robot control)

PLC address	Script address (Get/SetCoils)	Register type	Function
0 0001	0	B it	start
0 0002	1	B it	stop
0 0003	2	B it	pause
0 0004	3	B it	Enable
0 0005	4	B it	Enable
0 0006	5	B it	Clear alarm
0 0007	6	B it	Enter drag and drop
0 0008	7	B it	Exit drag and drop
0 0009	8	B it	Switch to automatic mode
0 0010	9	B it	Switch to manual mode
0 00 11~01024	10 ~1023	B it	Reserved bits
0 10 25~09999	10 24~9998	B it	User -defined

3 Contact register definition (map1, robot state)

PLC address	Script address (GetInBits)	Register type	Function
1 0001	0	B it	Running status
1 0002	1	B it	Stopped state
1 0003	2	B it	Paused
1 0004	3	B it	Safety origin state

1 0005	4	B it	Safe skin pause stat e
1 0006	5	B it	Idle state
1 0007	6	B it	-on state
1 0008	7	B it	Enable state
1 0009	8	B it	Alarm status
1 0010	9	B it	Collision state
1 0011	1 0	B it	drag state
1 0012	1 1	B it	Recovery mode st atus
1 00 13~19999	12 ~9998	B it	Undefined

4 Input register definition (map1, real-time feedback data from the robot)

registers 30001 ~30999 and 31722~39999 are undefined (except for 32001~32067, which stores the controller SN code, body SN code, and security checksum characters of type U1 6). The byte order is little-endian reversed .

PLC address	Data types	number	byte size	Script address (GetInRegs)	type	Function
3 1000	unsigned sh ort	1	2	9 99	U 16	Data Validity
3 1001	unsigned sh ort	1	2	1,000	U 16	Total message l ength in bytes
3 10 02~31004	-	-	-	10 01~1003	-	Reserved bit s
3 10 05~31008	ui nt64	1	8	10 04~1007	U 64	For numeric inp ut, please see [li nk/details]. DI /DO Instruc tions

3 10 09~31012	ui nt64	1	8	10 08~1011	U 64	Digital output, see details DI /DO Instructions
3 10 13~31016	ui nt64	1	8	10 12~1015	U 64	For details on Robot Mode, please see RobotMode. illustrate
3 10 17~31020	ui nt64	1	8	10 16~1019	U 64	Unix timestamps (in milliseconds)
3 10 21~31024	-	-	-	10 20~1023	-	Reserved bits
3 10 25~31028	ui nt64	1	8	10 24~1027	U 64	Memory structure test standard value 0x0123 4567 89AB CDEF
3 10 29~31032	-	-	-	10 28~1031	-	Reserved bits
3 10 33~31036	do uble	1	8	10 32~1035	F 64	Speed ratio
3 10 37~3104	-	-	-	10 36~1039	-	Reserved bits
3 10 41~31044	do uble	1	8	10 40~1043	F 64	Control board voltage
3 10 45~31048	do uble	1	8	10 44~1047	F 64	Robot voltage
3 10 49~31052	do uble	1	8	10 48~1051	F 64	Robot current
3 10 53~31056	do uble	1	8	10 52~1055	F 64	Script running status
3 1057	unsigned sh ort	1	2	1 056	U 16	Safe I/O input status
3 1058	unsigned sh ort	1	2	1 057	U 16	Safe I/O output status
3 10 59~31096	-	-	-	10 58~1095	-	Reserved bits

3 10 97~31120	do uble	6	4 8	10 96~1119	F 64	Target joint pos ition
3 11 21~31144	do uble	6	4 8	11 20~1143	F 64	Target joint vel ocity
3 11 45~31168	do uble	6	4 8	11 44~1167	F 64	Target joint acc eleration
3 11 69~31192	do uble	6	4 8	11 68~1191	F 64	Target joint cur rent
3 11 93~31216	do uble	6	4 8	11 92~1215	F 64	Target joint tor que
3 12 17~31240	do uble	6	4 8	12 16~1239	F 64	Actual joint pos ition
3 12 41~31264	do uble	6	4 8	12 40~1263	F 64	Actual joint vel ocity
3 12 65~31288	do uble	6	4 8	12 64~1287	F 64	Actual joint cur rent
3 12 89~31312	do uble	6	4 8	12 88~1311	F 64	TCP sensor force value (v ia six- dimensional force gauge) Calculate)
3 13 13~31336	do uble	6	4 8	13 12~1335	F 64	TC P Cartesia n actual c oordinate s
3 13 37~31360	do uble	6	4 8	13 36~1359	F 64	TC P Cartesia n actual v elocity val ue
3 13 61~31384	do uble	6	4 8	13 60~1383	F 64	TCP force va lue (Calculated usi ng joint cu rrent)
3 13 85~31408	do uble	6	4 8	13 84~1407	F 64	TC P Cartesia n target c oordinats

3 14 09~31432	do uble	6	4 8	14 08~1431	F 64	TC P Cartesia n target v elocity val ue
3 14 33~31456	do uble	6	4 8	14 32~1455	F 64	Joint temper ature
3 14 57~31480	do uble	6	4 8	14 56~1479	F 64	Joint control modes: 8: Position mode; 10 : Torque mode.
3 14 81~31504	do uble	6	4 8	14 80~1503	F 64	Joint voltage
3 15 05~31506	-	-	-	15 04~1505	-	Reserved bit s
3 1507	c har	1	1	1 5 06 Low B yte	I8	User coordina te system
3 1507	c har	1	1	1 5 06 high b ytes	I8	Tool coordinat e system
3 1508	c har	1	1	1 5 07 Low B yte	I8	Algorithm que ue runni ng flag
3 1508	c har	1	1	1 5 07 high b ytes	I8	Algorithm que ue paus e flag
3 1509	c har	1	1	1 5 08 low by te	I8	Joint velocity r atio
3 1509	c har	1	1	1 5 08 high b ytes	I8	Joint acceleratio n ratio
3 1510	c har	1	1	1 5 09 Low B yte	I8	Joint accelerat ion ratio
3 1510	c har	1	1	1 5 09 high b ytes	I8	Cartesian posi tion-velo city ratio
3 1511	c har	1	1	1 5 10 low by tes	I8	Descartes attit ude velo city ratio

3 1511	c har	1	1	1 5 10 high b ytes	I8	Cartesian posi tion accel eration rat io
3 1512	c har	1	1	1 5 11 low by te	I8	Cartesian attit ude accel eration rat io
3 1512	c har	1	1	1 5 11 high b ytes	I8	Cartesian posi tion accele ration ratio
						Descartes postu re plus
						Speed ratio
3 1513	c har	1	1	1 5 12 high b ytes	I8	on the robot 's brake st atus , plea se see [lin k/referenc e]. Bra keStatus Descrip tion
3 1514	c har	1	1	1 5 13 low by tes	I8	Robot Enable St ate
3 1514	c har	1	1	1 5 13 high b ytes	I8	Robot drag stat e
3 1515	c har	1	1	1 5 14 low by tes	I8	Robot motion st ate
3 1515	c har	1	1	1 5 14 high b ytes	I8	Robot alarm stat us
3 1516	c har	1	1	1 5 15 low by tes	I8	Robot jogging st ate
3 1516	c har	1	1	1 5 15 high b ytes	I8	For machine typ e details, please see [link/refere nce]. Rob otType Explan ation
3 1517	c har	1	1	1 5 16 low by tes	I8	Button panel dra g signal

3 1517	c har	1	1	1 5 16 high b ytes	I8	Button panel en able signal
3 1518	c har	1	1	1 5 17 low by tes	I8	Button panel rec ording signal
3 1518	c har	1	1	1 5 17 high b ytes	I8	Button panel sig nal reproduction
3 1519	c har	1	1	1 5 18 low by tes	I8	Button panel g ripper co ntrol sign al
3 1519	c har	1	1	1 5 18 high b ytes	I8	Six- dimensional force online sta tus
3 1520	c har	1	1	1 5 19 low by tes	I8	Collision stat e
3 1520	c har	1	1	1 5 19 high b ytes	I8	(Safety skin) Forearm near pause state
3 1521	c har	1	1	1 5 20 low by tes	I8	(Safety Skin) J4 is nearing a standstill.
3 1521	c har	1	1	1 5 20 high b ytes	I8	(Safety Skin) J5 is nearing a standstill.
3 1522	c har	1	1	1 5 21 low by tes	I8	(Safety Skin) J6 is nearing a standstill.
3 1522	c har	1	1	1 5 21 high b ytes	I8	Reserved bit s
3 15 23~31552	-	-	-	15 22~1551	-	Reserved bit s
3 15 53~31556	do uble	1	8	15 52~1555	F 64	Reserved bit s
3 15 57~31560	ui nt64	1	8	1556 ~1559	U 64	Current Algorith m Queue ID
3 15 61~31584	do uble	6	4 8	1560 ~1583	F 64	Actual torque

3 15 85~31588	do uble	1	8	1584 ~1587	F 64	Load weight (kg)
3 15 89~31592	do uble	1	8	15 88~1591	F 64	X- direction eccentricit y distance (mm)
3 15 93~31596	do uble	1	8	15 92~1595	F 64	Y- direction eccentricit y distance (mm)
3 15 97~31600	do uble	1	8	15 96~1599	F 64	Z -direction eccentricit y distance (mm)
3 16 01~31624	do uble	6	4 8	16: 00~1623	F 64	User coordina tes
3 16 25~31648	do uble	6	4 8	16 24~1647	F 64	Tool coordinat e values
3 16 49~31652	do uble	1	8	1648 ~1651	F 64	Track Reprodu ction Ru nning In dex
3 16 53~31676	do uble	6	4 8	1652 ~1675	F 64	Current raw v alues of si x-dimensi onal force data
3 16 77~31692	do uble	4	3 2	16 76~1691	F 64	[qw , qx, qy, qz] target quaternion
3 16 93~31708	do uble	4	3 2	16 92~1707	F 64	[qw , qx, qy, qz] are actual quaternions
3 1709	unsigned sh ort	1	2	1 708	U 16	Manual/Auto matic Mode 1: Manual 2: Automatic mode 0 : Mode not enabled Switch

3 1710	unsigned sh ort	1	2	1 709	U 16	USB drive exp ort status
3 1711	c har	1	1	1 7 10 low by te	I8	safe status
3 1711	c har	1	1	1 7 10 high b ytes	I8	Safety state res erved bit
3 17 12~31721	-	-	-	17 11~1720	-	Reserved bit s
			1 440			1440 bytes in total

PLC address	Script address (GetInRe gs)	type	Function
3 20 01~32032	2000 ~2031	U 16	The first to the 32nd characters of the controller 's serial number .
3 20 33~32065	20 32~2064	U 16	The first to the 32nd characters of the serial number.
3 2066	2 065	U 16	(2 bytes) of the " security checksum"
3206 7	206 6	U1 6	(2 bytes) of the " security checksum" are...

Explanation of motion parameter feedback values

If motion parameters (speed, acceleration, etc.) are set separately in the project, the relev ant feedback values will not be updated immediately, but will only be updated when the r obot executes the next motion command.

DI /DO Instructions

DI and DO occupies 8 bytes, with each byte containing 8 bits (binary), and can represent the status of up to 64 ports on each DI/DO. Each bit in each byte, from low to high, repr esents the status of a terminal, with 1 indicating that the corresponding terminal is ON, a nd 0 indicating that the corresponding terminal is OFF or there is no corresponding termi nal .

if the first byte of DI is 0x01, its binary representation is 00000001, and from low to high, they represent D1._____1 ~ DI The state _____of 8 , i.e. , DI_____1 is ON, and the other 7 DIs are OFF;

The second byte is 0x02, which is represented in binary as 00000010. From low to high, these represent D1._____9 ~ DI The state of ___16 , i.e., DI_____1. 0 is ON, and the other 7 DIs are OFF;

Subsequent bytes follow the same pattern. The number of I/O terminals varies dependin g on the control cabinet. Any binary bits exceeding the number of I/O terminals will be fill ed with 0.

RobotMode Explanation

Value	definition	illustrate
1	R O BOT_____MODE _I N IT	Initialization state
2	R O BOT_____MODE _BR AK E_____O PEN	with any joint
3	R O BOT_____MODE _P O WEROFF	robotic arm power-off state
4	R O BOT_____MODE _D IS ABLED	Not enabled (no brake released)
5	R O BOT_____MODE _E N ABLE	Enable and idle
6	R O BOT_____MODE _B AC KDRIVE	drag and drop mode
7	RO BOT_____MODE _R U NNING	Running status (project status, TCP que ue activity, etc.)
8	RO BOT_____MODE _S IN G LE_____M OVE	Single motion state (jog, RunTo, etc.)
9	RO BOT_____MODE _E RROR	There are unresolved alarms. This state has the highest priority; regardless of the robotic arm's current state, it will retur n 9 when an alarm occurs.

1 0	RO BOT_____MODE _P AUSE	Project suspended
1 1	R O BOT_____MODE _C OL LISION	Collision detection triggered stat e

Br akeStatus Description

This byte represents the brake status of each joint bit by bit, with a 1 indicating that the brake of that joint has been released. The correspondence between the number of bits a nd the joints is shown in the table below:

Numb er of digits	7	6	5	4	3	2	1	0
meani ng	Reserve d bits	Reserve d bits	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6

Example :

- 0x01 (00000001): Joint 6 brake released
- 0x02 (00000010): Joint 5 brake released
- 0x03 (00000011): Brakes on joints 5 and 6 released.
- 0x04 (00000100): Joint 4 brake released

RobotType Explanation

Value	Representative models
3	C R3
5	C R5
7	C R7
1 0	C R10
1 2	C R12
1 6	C R16
1 01	Nova 2
1 03	Nova 5
1 13	C R3A
1 15	C R5A
1 17	C R7A
1 20	R10A

1 22	With R12A
1 26	C R16A
1 30	R20A
1 50	Magician E6

5 Hold register definitions (map1, robot PLC interaction)

PLC address	Script address (Get/SetHoldRegs)	Register type	Function
4 00 01~41024	0 ~ 1023	-	Reserved bits
4 10 25~49999	10 24~9998	-	User -defined

Appendix B Block Programming Block Instructions

General Instructions

Block type

In block programming, blocks can be divided into four main categories based on their shape.

Round-headed building blocks

This type of block is used at the head of a block column, and all blocks to be executed m ust be placed below this type of block. Only event block groups contain this type of block.

rectangular blocks

This type of block is used to execute instructions and is a major component of the block program. The oval and rhomboid parameter slots in the block can be filled with blocks of the corresponding shapes.

A variant of rectangular blocks allows for nesting other

blocks within them, used for flow control. Example :

Elliptical building blocks

the elliptical block will return a variable (number/string/array, etc.), which can be placed i nto the parameter slot of the corresponding shape of the rectangular block as a paramet er.

example :

Rhombus building blocks

The diamond-shaped block is used for conditional judgment. After execution, it will retur n true or false and be placed into the parameter slot of the corresponding shape of the r ectangular block as a parameter.

example :

Exercise

The motion modes supported by robotic arms can be categorized as follows.

Joint movement

The robotic arm plans the motion of each joint based on the difference between the curre nt joint angles and the joint angles at the target point, enabling all joints to move simultan eously. Joint motion is not constrained by TCP (Tool). Center The trajectory of a Point is g enerally not a straight line.

Joint movement is not restricted by singular positions (see the corresponding hardware m anual for singular position details). Therefore, if there are no requirements for the moveme nt trajectory , or if the target point is near a singular position, it is recommended to use joi nt movement.

linear motion

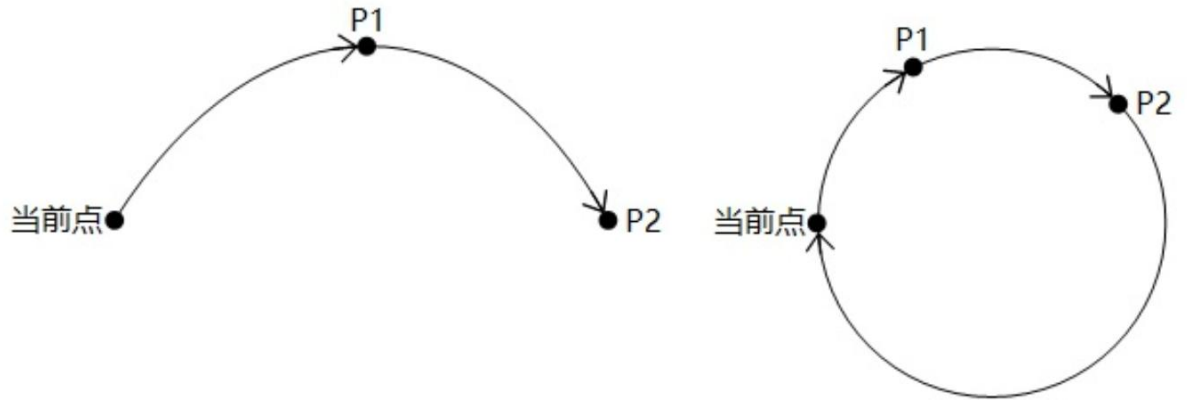
The robotic arm plans its motion trajectory based on the current pose and the pose of the target point, making the TCP motion trajectory a straight line, and the end effector's pos e changes at a constant speed during the motion.

When the movement trajectory passes through an odd position, issuing a linear motion co mmand to the robotic arm will generate an error. It is recommended to replan the position or use joint motion near the odd position .

Arc motion

The robotic arm determines an arc or a complete circle using its current position and thr ee non-collinear points, P1 and P2 . The end effector posture of the robotic arm during t he movement is calculated by interpolating the postures of the current point and P2.

The posture of point P1 is not included in the calculation (i.e., the posture of the robotic arm when it reaches point P1 during the movement may be different from the taught posture).



When the motion trajectory passes through an odd position, issuing an arc motion command to the robotic arm will generate an error. It is recommended to replan the position or use joint motion near the odd position .

Coordinate system parameters

The motion blocks can specify the user coordinate system and tool coordinate system corresponding to a point through advanced configuration. The parameter priority is as follows:

1. through advanced configuration, the specified coordinate system will be used. If the point parameter is a teaching point, the pose coordinates of the teaching point will be converted to values in the specified coordinate system before use.
2. is not specified through advanced configuration, and the point parameter is a teaching point, the coordinate system index provided by the teaching point will be used.
3. is not specified through advanced configuration, and the point parameter is a joint variable or pose variable, the global coordinate system set in **the control block group will be used (see the Set User Coordinate System** and **Set Tool Coordinate System** blocks for details; the default coordinate system when not using block settings is 0).

illustrate :
starts running , the default global coordinate system is set to 0, regardless of the value set in the jog panel before running the project.

Speed parameters

relative speed

The motion block can be configured with advanced settings to specify the acceleration (Accel) and velocity ratio (V) when the robotic arm executes the motion command.

```
actual movement speed of the robotic arm = Maximum speed x
Global Rate x Command rate robotic arm actual motion acceler
ation = maximum acceleration x Instruction rate
```

The maximum velocity/acceleration is controlled by **the reproduction parameters** , which can be found in CBSH Studio . View and modify **motion parameters on the** Pro page.

Global speed can be achieved through CBSH Studio Pro's speed adjustment slider (top right corner of the image above) or SpeedFactor command settings.

The command rate is the rate specified in the advanced configuration of the motion bloc ks. If the motion acceleration/velocity ratio is not specified in the advanced configuration, the corresponding **joint/line is used by default. Velocity/Acceleration** The default value f or **the scale block setting is 100 when no block settings are used.**

Absolute speed

linear and curved motion blocks allow you to specify the absolute speed (Speed) when the ro botic arm executes the motion command.

The absolute speed is not affected by the global speed, but is limited by the maximum spe ed in **the reproduction parameters** (if the robotic arm enters the reduction mode, it is limit ed by the maximum speed after reduction). That is, if the absolute speed is greater than

the maximum speed in the reproduction parameters, the maximum speed shall prevail.

For example, if the absolute speed of linear motion is set to 1000, which is less than the maximum speed of 2000 in the reproduction parameters, the robotic arm will move at... The target speed is 1000 mm /s, which is independent of the global velocity at this point. However, if the robotic arm is in reduced mode... If we assume a reduction rate of 10%, the maximum speed becomes 200, which is less than 1000. At this point, the robotic arm will move at a target speed of 200 mm/s.

Absolute speed and speed ratio cannot be set simultaneously.

Smooth transition

In some cases , a robotic arm needs to pass through multiple transition points before reac hing its target point. These transition points are typically used to allow the robot's moveme nt path to avoid obstacles, and the robotic arm does not need to reach these points precis ely. By setting smooth transition parameters, the robotic arm will begin turning before reac hing the transition points , making the speed and trajectory of the robotic arm's turns smo other.

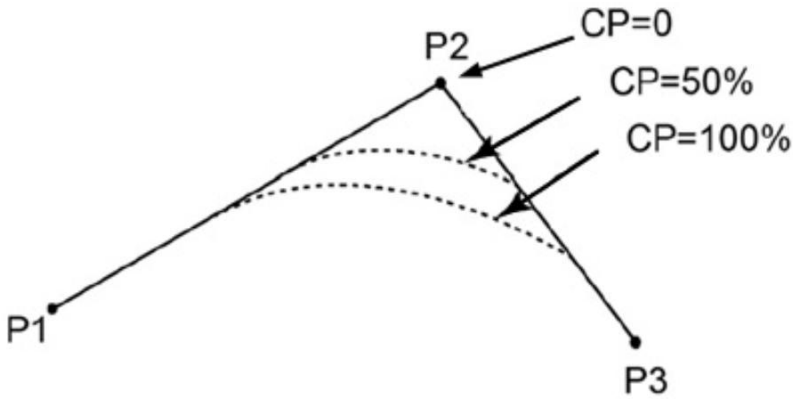
Users can specify the smooth transition ratio (CP) or smooth transition radius (R) when the robotic arm transitions from one motion command to the next through advanced motion command configuration .

illustrate :

- The joint motion mode does not support setting a smooth transition radius (R).
- If the several path points specified by the user are based on different tool coordinate sy stems, a smooth transition cannot be achieved.

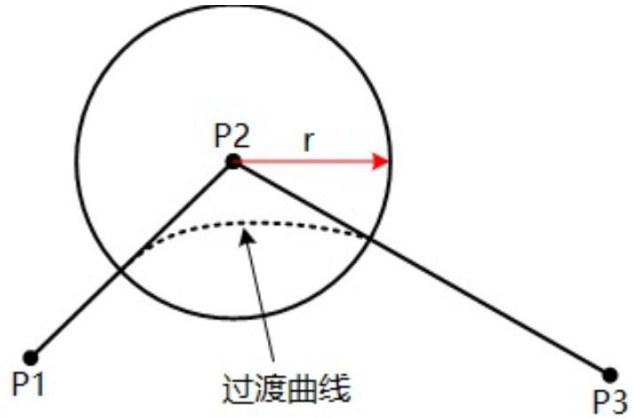
C P

setting the smooth transition ratio, the system automatically calculates the curvature of the transition curve. The larger the CP value, the smoother the curve, as shown in the figure below. The CP transition curve is affected by the motion speed/acceleration. Even if the location and CP value are the same, the curvature of the transition curve will be different when the motion speed/acceleration is different.

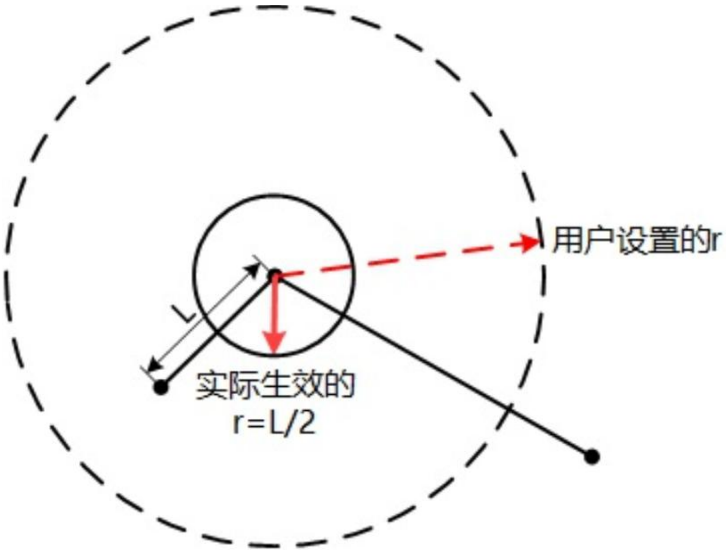


R

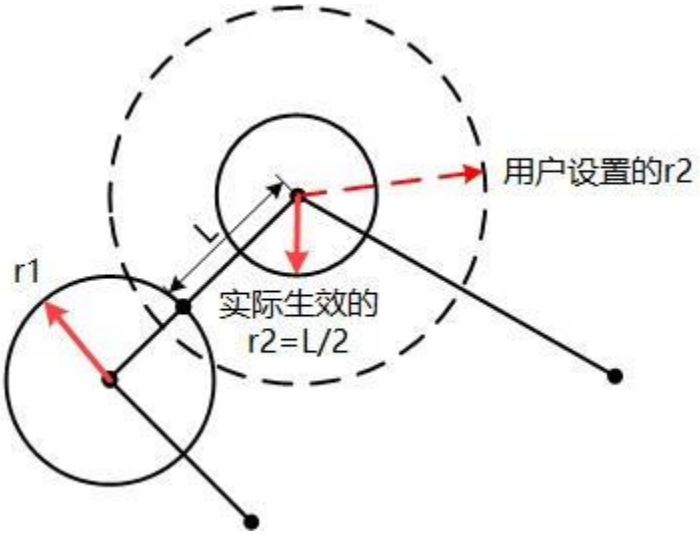
setting a smooth transition radius, the system calculates a transition curve with the transiti on point as the center and the specified radius. The R-transition curve is not affected by motion speed /acceleration, but is determined only by the point location and the transitio n radius.



If the user -set transition radius is too large (exceeding the distance between the starting point/end point and the transition point), the system will automatically use half of the shor ter distance between the starting point/ end point and the transition point (L in the figure below) as the transition radius to calculate the transition curve.



When two consecutive transition radii (r1 and r2 in the figure below) coincide, the system will take the point after the first motion transition is completed as the starting point of the second motion, and then calculate the second r that actually takes effect according to the logic of the transition radius being too large.



default value

smooth transition ratio or radius is specified in the advanced settings, the default value of the **smooth transition ratio block setting will be used.** The default value is 0 when no block setting is used.

illustrate :

A smooth transition will cause the robotic arm to move without passing through the midpoint. Therefore, when a smooth transition is set, the IO signal output or function setting (such as switching on and off the safety skin) between two motion commands will be executed during the transition.

If you want the command to be executed when the robotic arm accurately reaches the midpoint, set the smooth transition parameter of the previous command to 0 .

Stop condition

Some motion blocks support specifying stopping conditions through advanced configurati on. If the specified stopping condition is met during the execution of motion instructions, th e robot will end the current motion and directly execute the next instruction.

- Supported conditions include IO, variables, and any expression that conforms to Lua synt ax.
- It supports up to 3 judgment conditions. Multiple conditions are connected by keyw ords **AND** (all conditions must be met) / **OR** (any one condition must be met) but d oes not support mixing **AND** and **OR** .

illustrate :

- Specifying a stopping condition will invalidate the smooth transition parameter r.
- specifying a stop condition, the smooth transition parameter cp will take effect, but the transition segment (curved portion) is not within the scope of the stop c ondition .
- If a smooth transition is added to the instruction preceding the stop condition, th e stop condition will not be triggered until the instruction leaves the transition s egment.

Event Blocks

Event blocks are used as markers to indicate when a program starts running; only the block fo llowing an event block will be executed.

Start running

Description: Identifier for the program's main thread. A running indicator will appear by de fault in the programming area after creating a new project. Please place other non-event blocks below it for programming.

Usage restrictions: Only one "Start Running" block can be used in a project.

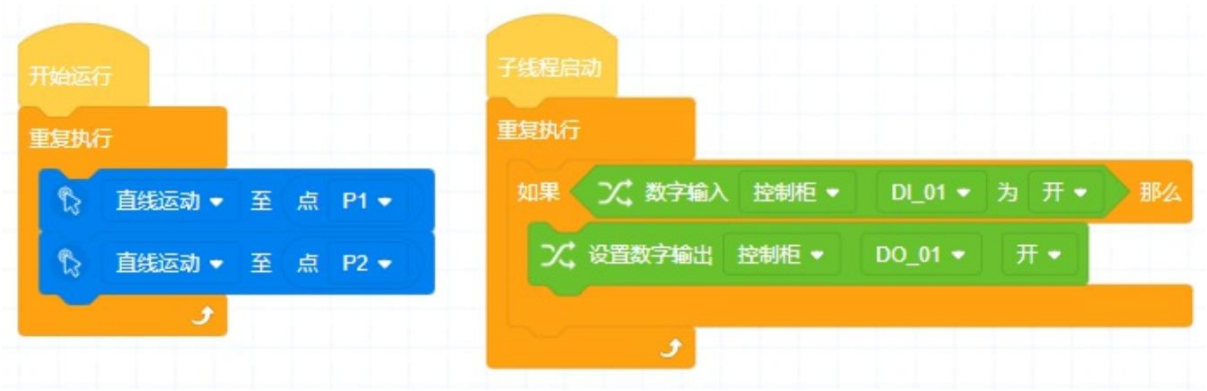
Startup of child threads

Description: Identifier for a program's child thread. Child threads run synchronously with t he main thread, but they cannot use robotic arm control commands; they can only perfor m variable calculations or I/O control, etc. Please choose whether to use child threads bas ed on the program's logic requirements.

Usage limitations: A project can have a

maximum of 4 child threads. **Example :**

In the following example, after the project is run, the main thread and the child thread will start running simultaneously. The motion instructions of the main thread and the I/O instructions of the child thread do not interfere with each other and are executed according to their respective queues.



Control block group

Control blocks are used to control the program's execution path.

Wait until the conditions are met

Description : The program pauses until the parameter is true before resuming execution.

Parameters : Use other hexagonal

blocks as parameters. **Example :**

The robotic arm moves to P1 and waits until DI1 is open before moving to P2.

Repeat n times

Description : Nest other blocks inside this block, and the nested block instructions will be exe cuted repeatedly a specified number of times.

Parameter : The

number of times to

repeat the execution.

Example 1:

The robotic arm repeatedly executes two motion commands 10 times.

Example 2:

The robotic arm executes two motion commands cyclically n times, where n is the value of th e numerical variable count.

Continuous repetitive execution

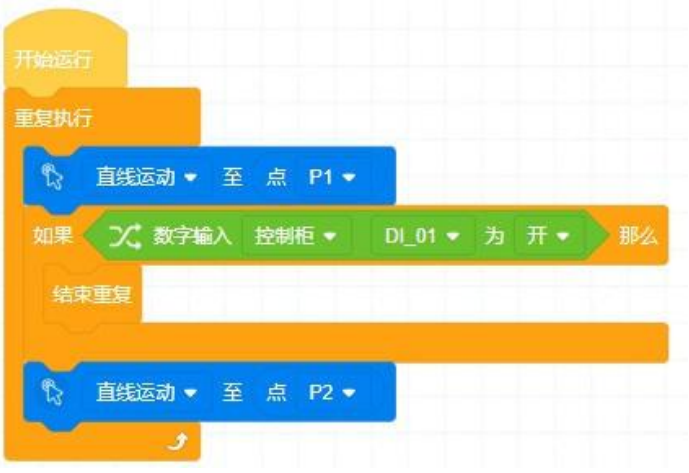
Description: Nest other blocks within this block. The nested block's instructions will be exe cuted repeatedly until the end-of-repetition block is encountered . No other blocks can b e placed under this block.

End repetition

Description: Used in blocks nested within the following repetitive execution classes. When the program reaches this block, the repetition will end directly, and the block instructions following the repetitive block will be executed.

Example :

repeated reciprocating motion of P1 and P2, if DI1 is on after the robotic arm moves to point P1, the repetition will immediately end.



Execute after the conditions are met

Description: If the parameter is true, the nested block instruction is executed. If the parameter is false, it jumps directly to the next block instruction. **Parameter:** Use other hexagonal blocks (returning a boolean value, either true or false) as parameters.

Execute accordingly if the conditions are met or not.

Description: If the parameter is true, execute the nested block instruction preceding "oth erwise". If the parameter is false, execute the nested block instruction following "otherw ise".

Parameters: Use other hexagonal blocks (returning a boolean value,

either true or false) as parameters. **Example :**

If DI1 is on, the robotic arm moves to point P1; otherwise, it moves to point P2.

Repeat until the condition is met.

Description : Repeatedly execute nested block instructions until the parameter is true.

Parameters: Use other hexagonal blocks (the return value is a boolean value, i.e., true or false) as parameters.

Set tags

Description : Set a label. After setting the label, you can

jump to the appropriate block using the label. **Parameter :**

Label name, which must start with a letter and cannot

contain spaces or other special characters.

Tag jump

Description : When the program reaches this block, it will jump directly to the specified label and execute the block instructions following the label.

Parameter : The set label

name. **Example :**

If DI1 is on, the robotic arm moves directly to P1; otherwise, the robotic arm moves to P2 first, and then to P1.

illustrate :
This block can only jump to the label block in the same column (including subroutin es), and cannot jump across columns. For example, it cannot jump from a user-def ined function to the main thread.

Instruction folding

Description: Allows nested blocks to be collapsed and displayed. It has no control functio n; it's only used to make the program more visually appealing and readable. Click the dou ble arrow in the lower right corner of the block to toggle the collapsed state.

Parameter : Describes the folded block; it is recommended to use an indirect and intuitive na me.

pause

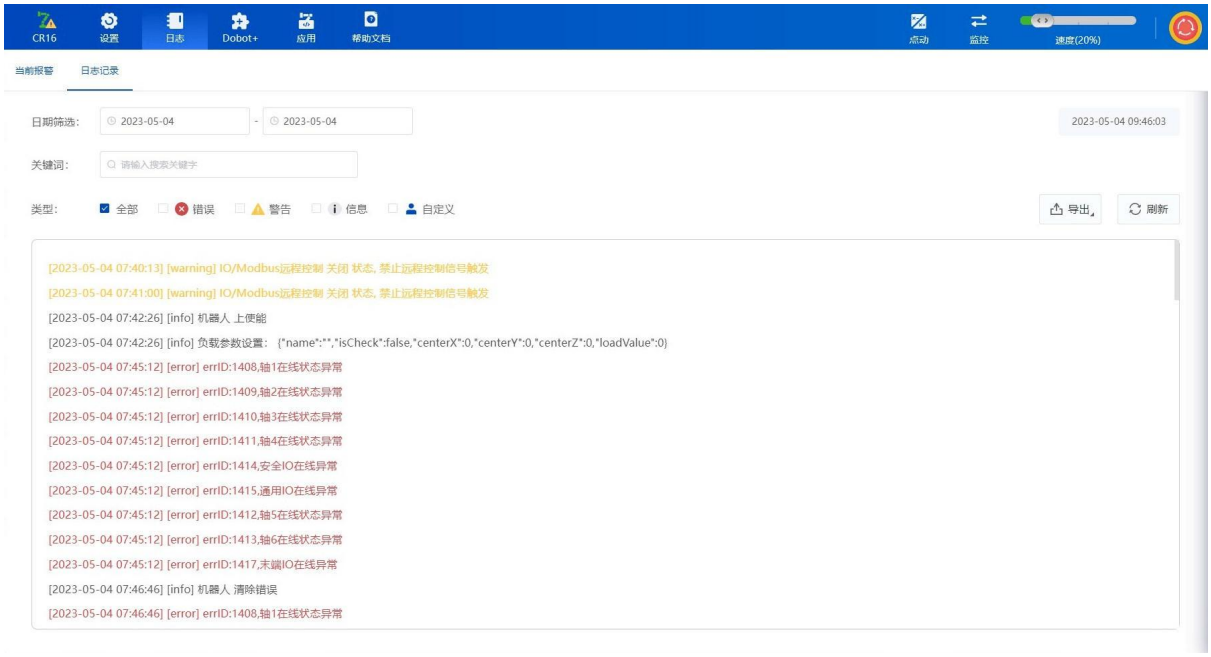
Description : The program pauses automatically after reaching this block and can only continue running through control software or remote control.

Stop program

Description : The program will automatically stop and exit after reaching this block.

Custom Logs

Description : Outputs custom-level log information, which can be viewed and exported from the control software's log page.



Configure collision detection function

Description: Sets up collision detection functionality. The collision detection level set using this block is only effective during project execution and will revert to the previous value after the project stops.

Parameters : Select the sensitivity of the collision detection function. You can choose to turn it off or select level 1 to 5. The higher the level number, the more sensitive the collision detection.

Set collision backoff distance

Description: Sets the distance the robotic arm will retract along its original path after detecting a collision. This block setting only applies during the current run of the project; it reverts to its original value after the project stops .

Parameter : Sets the collision backoff distance, value range: [0, 50], unit: mm

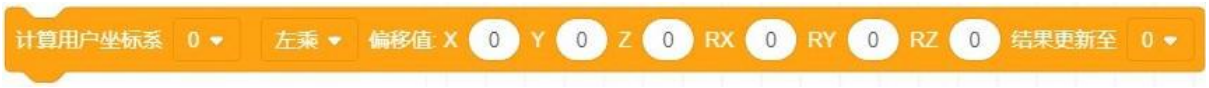
Modify user or tool coordinate system

Description : Modifies the specified user or tool coordinate system.
parameter :

- Specify whether to modify the user coordinate system or the tool coordinate system.
- Specify the number of the user coordinate system to be modified.
- Specify the modified user coordinate system parameters.
- Specify the scope of the changes:
  - Script Only : The coordinate system modified by this command only takes effect during the current project's operation and will revert to its original value after the project stops .
  - Save globally : The coordinate system modified by this command is saved globally, and the modified values will still be retained even after the project stops.

Example :

Calculate and update the user coordinate system



Description: Calculates and updates the specified user coordinate system. This modificati on only takes effect during the current project run; the coordinate system will revert to its previous value when the project stops.

parameter :

- Specifies the number of the user coordinate system to be used as the calculation referen ce. User coordinate system 0 is initialized as the base coordinate system.
- Specify the calculation direction.
  - Left multiplication: indicates that the coordinate system specified by the previous para meter is deflected along the base coordinate system.
  - Right multiplication: indicates that the coordinate system specified by the previous par ameter rotates along itself.
- Specifies the offset value of the coordinate system.
- The system will calculate a new user coordinate system using the above parameters. The user needs to specify which user coordinate system the calculation results shoul d be updated to.

Example :

- Calculate the left offset value for user coordinate system **1 : X 10** Y **10** Z **10** RX **10** RY **10** R Z **10.** Update the result to... **1**

  The above command indicates that an initial pose is in a coordinate system identical to user coordinate system 1, and the coordinate system is translated along the base coordinate system {x=10, y=10, z=10} and rotate {rx=10, ry=10, After rz=10}, the new coordinate system is assigned to user coordinate system 1.

- Calculate the offset value to the right in user coordinate system **1** : X **10** Y **10** Z **10** RX **10** RY **10** RZ **10.** Update the result to... **1**

The above command indicates that an initial pose is in a coordinate system identical to user coordinate system 1, and the coordinate system is translated along user coordinate system 1 by {x=10, } y=10, z=10} and rotate {rx=10, ry=10, After rz=10}, the new coordinate system is assigned to user coordinate system 1.

Calculate and update the tool coordinate system

Description: Calculates and updates the specified tool coordinate system. This modificatio n only takes effect during the current project run; the coordinate system will revert to its p revious value when the project stops.

parameter :

- Specifies the number of the tool coordinate system to be used as the calculation referenc e. Tool coordinate system 0 is initially set to the flange coordinate system.
- Specify the calculation direction.
  - Left multiplication: indicates that the coordinate system specified by the previous para meter is deflected along the flange coordinate system.
  - Right multiplication: indicates that the coordinate system specified by the previous par ameter rotates along itself.
- Specifies the offset value of the coordinate system.
- The system will calculate a new tool coordinate system using the above parameters. The user needs to specify which tool coordinate system to update the calculation res ults to.

Example :

- Calculation tool coordinate system **1, left multiplied by** offset value: X **10** Y **10** Z **10** RX **10** RY **10** RZ **10** Results updated to **1**

  The above command indicates an initial pose in a coordinate system identical to tool coordinate system 1, and a translation along the flange coordinate system {x=10, y=10, z=10} and rotate {rx=10, ry=10, After rz=10}, the new coordinate system is assigned to the tool coordinate system 1.

- Calculation tool coordinate system **1, right multiplied by** offset: X **10** Y **10** Z **10** RX **10** RY **10** RZ **10** Results updated to **1**

  The above command indicates an initial pose in a coordinate system identical to tool coordinate system 1, and a translation of {x=10, } along tool coordinate system 1. y=10, z=10} and rotate {rx=10, ry=10, After rz=10}, the new coordinate system is assigned to the tool coordinate system 1.

Set user coordinate system

Description: Sets the user coordinate system used by the current project. It reverts to its original value when the project stops.

Parameter : User coordinate system index.

Set tool coordinate system

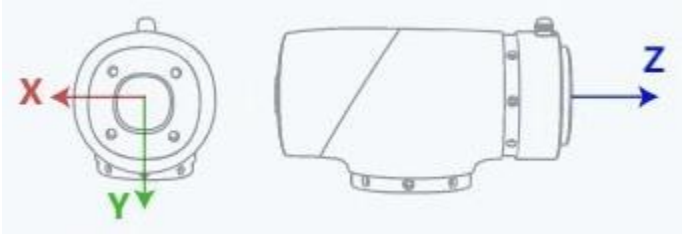
Description: Sets the tool coordinate system used in the current project. It reverts to its original value when the project stops.

Parameter : Tool coordinate system index.

Set load parameters

Description: Sets the weight and eccentricity coordinates of the end load; these values revert to their original values after the project stops. **Parameters :**

- Enter the current load weight. Unit: kg
- Enter the eccentric coordinates of the current load. Refer to the diagram below for coordinate axis directions. Unit: mm



Wait for the specified time

Description : Wait for a specified time before executing the next instruction.

Parameter: The delay time for issuing the instruction . The maximum waiting time is 214 7483647ms. Setting the parameter beyond the maximum value will cause the instructio n to be invalid.

Install a safety wall switch

Description : Sets the switch for a single safety wall; the

value reverts to its original state after the project stops.

Parameters :

- Select the security wall index. A currently selected security wall will be displayed as a que stion mark if it is deleted.
- Select the status of the safety wall switch.

Description : Sets the on/off switch for a single safety

zone; the value reverts to its original state after the project

stops. **Parameters :**

- Select the safe zone index. A question mark will appear if the currently selected safe zone is deleted.
- Select the safety zone switch status.

Get system time

Description : Get the current system time.

Return value: The system's current Unix timestamp, converted to milliseconds, i.e., from 00:00 GMT on January 1, 1970. The number of milliseconds up to the current time is generally used to calculate time differences. To obtain the local time, please convert it using the obtained Greenwich Mean Time (GMT) according to your local time zone .

Example :

If the obtained value is 1686304295963, the corresponding Beijing time is 2023-06-09. 17:51:35 (plus 963 milliseconds); If the obtained value is 1686304421968, it translates to 2023-06-09 Beijing time. 17:53:41 (plus 968 milliseconds); the time difference can be calculated by subtracting the values obtained multiple times .

Start timing

Description : The timer starts when the project reaches this block. It needs to be used in conjunction with the block below that **retrieves the timing result** .

Get timing results

Description : End the timer and return the time difference.

Return value: The time difference from the start to the end of the timer, in milliseconds. The maximum countable time is 4,294,967,295 ms (approximately 49.7 days) . After this time, the count will restart from 0.

Example :

the printing robot to move from joint P1 to joint P2.

Set End Tool Mode

Description: For models with AI1 and AI2 interfaces on the robotic arm end effector that use multiplexed terminals with the 485 interface (CR and CR...). (Series A) This interface allows you to set the mode of the end multiplexed terminal. The default mode is 485 mo de.

> **illustrate :**
> that do not support end-effector mode switching.

parameter :

- Select the terminal mode. Supports both 485 mode and analog input mode. In 485 mode, the following two parameters do not need to be set.
- Select the analog input mode for AI1. Supports 0~10V voltage input mode, current mode, and 0~5V voltage input mode.
- Select the analog input mode for AI2. Supported modes are the same as for AI1.

Set the terminal 485 data format

Description : Sets the data format for the

RS485 interface of the terminal tool.

Parameters :

- Enter the baud rate for the RS485 interface.
- Choose whether to include a parity bit.
- Select the length of the stop bits.

Set the end power switch

Description: Sets the power supply status of the end effector, typically used to restart the end effector power supply, such as re-powering and initializing the end effector gripper. I f this interface is to be called continuously, it is recommended to wait at least 4ms betwe en calls.

> **illustrate :**
> The terminal DO will also fail after the terminal power is turned off .

Parameter : Select the power switch status.

Custom pop-up window

Description: Used to display a window with user-customizable information when the scr ipt is running. CBSH Studio Pro will only display one pop -up at a time , containing the l atest pop-up information.

If CBSH Studio is not open when the project runs. In Pro mode (e.g., when starting the proj ect via IO or Modbus), no pop-up window will appear , but warning and error pop-up com mands will still cause the project to pause.

parameter :

- Message type.
  - Information-type pop-ups: These pop-ups do not affect project operation. Clickin g " **Stop**" will halt the project, and clicking **"OK** " will close the pop-up directly. Us ers can also stop the project via IO or Modbus after the pop -up appears; the po p-up will disappear automatically after the project stops.

  - Warning type : This type of pop-up will pause the project. Click **Stop** to stop the project, and click **Continue** to continue the project. After the pop-up appears , th e user can also stop/continue the project via IO or Modbus. The pop-up will disa ppear automatically after the project stops/continues.

  - Error type : This type of pop-up will pause the project. Click **Stop** to stop the proj ect, and click **Continue** to continue the project. After the pop-up appears , the us er can also stop/continue the project via IO or Modbus. The pop-up will disappea r automatically after the project stops/continues.

- `The title of the pop-up` to be displayed. Only strings are support ed, and the length cannot exceed 128.
- `The information` to be displayed . Various variables, including strings, are supported. If it's a string, the length cannot exceed 128 bytes; if it's another vari able type, it will be converted to a character, and the size of the converted character set cannot exceed 512 bytes.

- Should the pop-up content be written to the log?
  - No : Do not write to the log.
  - Yes: Write the corresponding type of log (where "Information" pop-up messages correspond to "Custom" type logs). The log content is "Pop-up title: Pop-up content".

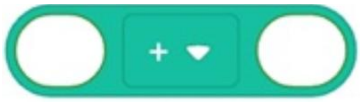
Notes

Description: Add a comment. Comments do not affect the execution of the project; they are mainly used to help those reviewing the project understand the project logic. **Parameter :** The content of the comment.

Operation Blocks

Operation blocks are used to perform operations on variables or constants.

Four arithmetic operations



Description : Performs

arithmetic operations on

the parameters.

Parameters :

- Fill in the variables or constants involved in the calculation on both sides. You can use an oval block that returns a numerical value or fill it in directly.
- Select one of the four arithmetic operators in the middle. **Return value:** The numerical value of the result of the operation.

Comparison operations

Description : Performs

comparison operations on

the parameters.

Parameters :

- Fill in the variables or constants involved in the calculation on both sides. You can use an oval block that returns a numerical value or fill it in directly.
- Choose the comparison operator in the middle.

Return value : Returns true if the comparison result is true, and false if the result is false.

AND operation

Description : Performs a bitwise AND operation on the parameters.

Parameters : Fill in the variables involved in the calculation on both sides, and use other hexa gonal blocks.

Return value : Returns true if both parameters are true, and false if either parameter is false.

OR operation

Description : Performs an OR operation on the parameters.

Parameters: Fill in the variables involved in the calculation on both sides, and use other hexagonal blocks. **Return value :** Returns true if either of the two parameters is true, and false if both are false.

NOT operation

Description : Performs a NOT operation on the parameter.

Parameters : Enter the variables to be used in the calculation, using other hexagonal blocks.

Return value: Returns false if the parameter is true, and true if it is false.

Modulo operation

Description : Performs a modulo operation on the parameter.

Parameters: Enter the variables or constants involved in the calculation on both sides. You can use an oval block that returns a numerical value or enter them directly.

Return value: The numerical value of the calculation result.

Rounding calculation

Description : Performs rounding operations on parameters.

Parameters: Enter the variables or constants involved in the calculation. You can use an oval block that returns a numerical value or enter them directly. **Return value:** The numerical value of the calculation result.

Single- value operations

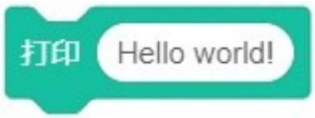
Description: Performs various single-value operations on parameters. The input values for trigonometric functions (sin/cos/tan) and the return values for inverse trigonometric functions (as in/acos/atan) are both angles.

parameter :

- Choose the calculation method.
  - absolute value
  - Round down
  - Round up
  - square root
  - sin
  - cos
  - tan
  - asin
  - acos
  - atan
  - ln
  - loh
  - e^
  - 10^
- Enter the variables or constants involved in the calculation. You can use an oval block that returns a numerical value or enter them directly.

Return value: The numerical value of the result of the operation.

Print



Description : Outputs parameters to the

console for viewing, primarily for debugging.

Parameters :

To output variables or constants to the console, you can use other elliptical blocks or fill them in directly.

Character blocks

The character blocks group contains commonly used functions for strings and arrays.

Get the nth character of a string

Description : Retrieves the nth character

of a string that is used as a variable.

Parameters :

- The first parameter should be a string; you can use other oval blocks or enter the string di rectly.
- The second parameter specifies which character in the string to return. **Return value:** The character at the specified position in the string.

Determine if string 1 contains string 2

Description : Determines whether the string of the first parameter contains the string of the s econd parameter.

Parameters: The two strings to be compared. You can use an elliptical block

that returns a string or fill them in directly. **Return value :** Returns true if string

one contains string two, otherwise returns false.

Concatenate two strings

Description: Concatenates two strings into one string, appending the second

string to the first. **Parameters:** The two strings to be concatenated. You can

use an oval block that returns a string or fill in the parameters directly. **Return**

value: The concatenated string.

Get the length of a string or array

Description: Gets the length of a specified string or array. The length of a string refers to t he number of characters in the string, and the length of an array refers to the number of elements in the array.

Parameters: The string or array whose length you want to calculate. You can

use an elliptical block that returns a string or array. **Return value:** The length

of the string or array.

Comparing two strings

Description : Compare the size of two strings based on their ASCII codes.

Parameters : The two strings to be compared. You can use an elliptical block that returns a st ring or fill it in directly.

Return value : Returns 0 if string 1 and string 2 are equal, -1 if string 1 is less than string 2, and 1 if string 1 is greater than string 2.

Convert array to string

Description: Converts a specified array to a string, with different array elements separated by a specified delimiter. For example, if the array is {1,2,3 } and the delimiter is |, the converted string will be "1|2|3".

parameter :

- To convert an array into a string, use an elliptical block that returns an array.
- The delimiter used

for conversion .

Return value: The converted string.

Convert a string to an array

Description: Converts a specified string into an array, separating the string ac cording to a specified delimiter. For example, if the string is "1 |2|3" and the delimiter is "|", the converted array will be {[1]=1,[2]=2,[3]=3}.

parameter :

- To convert a string to an array, you can use an elliptical block that returns a string or input it directly.
- The delimiter used for the conversion.

Return value: The converted array.

Get the element at a specified index in the array

Description: Retrieves the element at a specified index in a given array. The index indicates the position of the element in the array; for example, the index of 8 in the array {7,8,9} is 2.

parameter :

- The target array is an elliptical block that returns an array.
- Specifies the index of the element.

Return value: The value of the element at the specified position in the array.

Get multiple specified elements from an array

Description: Retrieves elements at multiple specified indices from a given array . Elements are retrieved within a range of start and end indices, based on a step value.

parameter :

- The target array is an elliptical block that returns an array.
- The range of elements to be retrieved is specified by the start and end indices.
- The step value determines the frequency of element retrieval: 1 retrieves all elements, 2 retrieves every other element, and so on. **Return value:** A new array of the specified elements.

Set the specified element in the array

Description : Sets the value of the

element at the specified index in the

array. **Parameters :**

- The target array is an elliptical block that returns an array.
- Specifies the index of the element.
- Specifies the value of the element.

Custom block sets

Custom block groups are used to create and manage custom blocks, as well as to access glo bal variables.

Calling global variables

Description : Calls global variables set in the control software.

Parameters: The name of the global variable to select. The currently

selected global variable will be displayed as a question mark if it is

deleted. **Return value:** The value of the global variable.

Set global variables

Description: Sets the specified variable. Note that the blocks for setting global variables a nd custom variables look the same, but their functionality differs slightly .

parameter :

- Select the name of the variable you want to modify. The currently selected global variable will be displayed as a question mark after it is deleted.
- The modified value can be entered directly or other oval blocks can be used.

Create a new custom variable

Click to create a new custom variable. The variable type can be either numeric or string. Variable names must begin with a letter and cannot contain spaces or other special chara cters. After creating at least one variable, the following custom variable-related blocks wil l appear in the block list.

Custom numerical variables

Description: This is a newly created custom numeric variable. Its default value is nil; it is r ecommended to assign a value before use. You can modify the name of the currently s elected variable or delete it using the variable selection dropdown .

Return value: The value of the variable.

Set the value of a custom numeric variable

Description: Sets a specified numerical variable. Note that the blocks for setting global var iables and custom variables look the same, but their functions differ slightly.

parameter :

- Choose the name of the variable you want to modify.
- The modified value can be filled in directly or other oval blocks can be used.

Adding or removing the value of a custom numeric variable

Description : Increments the

specified numeric variable by the

specified value.

Parameters :

- Choose the name of the variable you want to modify.
- To increase a value, you can either fill it in directly or use other oval blocks. Setting it to a negative number will decrease the value.

Custom string variables

Description: A newly created custom string variable with a default value of nil. It is reco mmended to assign a value before use. You can modify the name of the currently selec ted variable or delete the variable using the variable selection dropdown.

Return value: The value of the variable.

Set the value of a custom string variable

Description : Sets the

specified string variable.

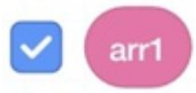
Parameters :

- Choose the name of the variable you want to modify.
- The modified value can be entered directly as a string.

Create an array

Click to create a new custom array. The array name must begin with a letter and cannot use spaces or other special characters. After creating at least one array , the following arr ay-related blocks will appear in the block list.

Custom array



Description: Newly created custom arrays are empty by default; it is recommended to assign values before use. Right-clicking (PC) / long-pressing (mobile) a block in the block list allows you to rename or delete the array. You can also rename or delete the currently selected array through the array selection dropdown menu in other array blocks . The checkboxes before array blocks are currently unavailable and can be ignored.

Return value: The values of the array.

Add variables to array

Description : Adds a variable to the specified array. The newly

added variable will become the last item in the array.

Parameters :

- The variables to be added can be filled in directly or other oval blocks can be used.
- Select the array you want to modify.

Delete specified item from array

Description : Deletes the specified item from the specified array.
parameter :

- Select the array you want to modify.
- The item number to be deleted can be entered directly or another oval block that returns a numerical value can be used.

Delete all items in the array

Description : Deletes all

items in the specified array.

Parameter : Select the array

to modify.

Inserting items into an array

Description : Inserts a variable at a

specified position in an array.

Parameters :

- Select the array you want to modify.
- The insertion position can be filled in directly or other oval blocks that return numerical val ues can be used.
- The variables to be added can be filled in directly or other oval blocks can be used.

Replace the specified item in the array

Description : Replaces a specified item

in an array with a specified variable.

Parameters :

- Select the array you want to modify.
- The number of the item to be replaced can be entered directly or another oval block that r eturns a numerical value can be used.
- The replaced variable can be filled in directly or other oval blocks can be used.

Get a specified item from an array

Description : Retrieves the

value of a specified item in

an array. **Parameters :**

- Select the array of items you want to retrieve.
- To retrieve the item number, you can either enter it directly or use another oval block that returns a numerical value. **Return value:** The value of the specified item.

Get the total number of items in the array

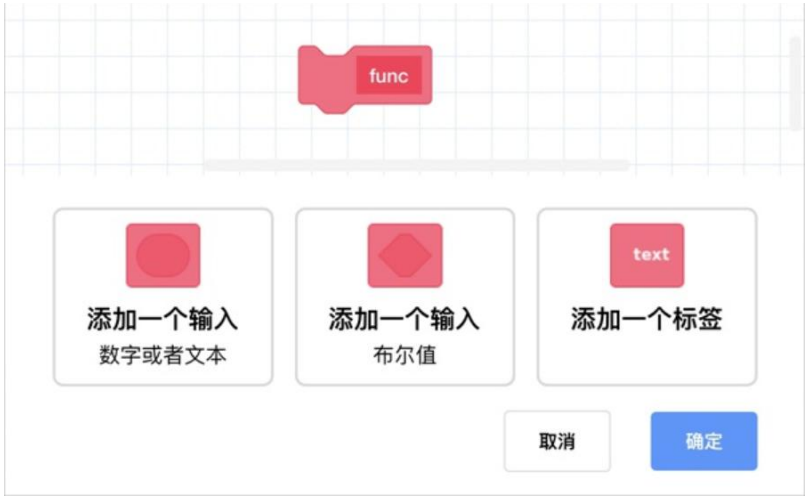
Description : Retrieves the total number of items in an array. **Parameter :** Select the array of items to retrieve.

Return value: The total number of items in the specified array.

Create a new function

Click to create a new function. A function is a fixed piece of code. You can define a group of blocks that implement a specific, frequently used function as a function . Then, each time you need to use that function, you only need to call this function, without having to repeatedly build the same group of blocks. Creating a new function requires declaring and defining it. After successfully creating a function, the corresponding function block will appear in the block list.

1. **Declare function**



This interface requires you to define the function name, and the types, number, and names of the input (parameters). Function and parameter names cannot contain spaces or other special characters. You can also add tags to the function; these tags can be used as comments to provide further information about the function or input. bright .

1. **Define function**

is completed , the function definition header block will appear in the programming area.

You need to connect blocks below this block to program and define the function's purpose.

the definition header can be dragged out and used in the blocks below, indicating that the inp ut used when the function is actually called is used as an argument.

User- defined functions

Description: User-defined function blocks. The name and input parameters are user-defin ed, used to call predefined functions . Right-click (PC) / long-press (mobile) a block in the block list to modify its declaration. To delete a function, delete its definition header block.

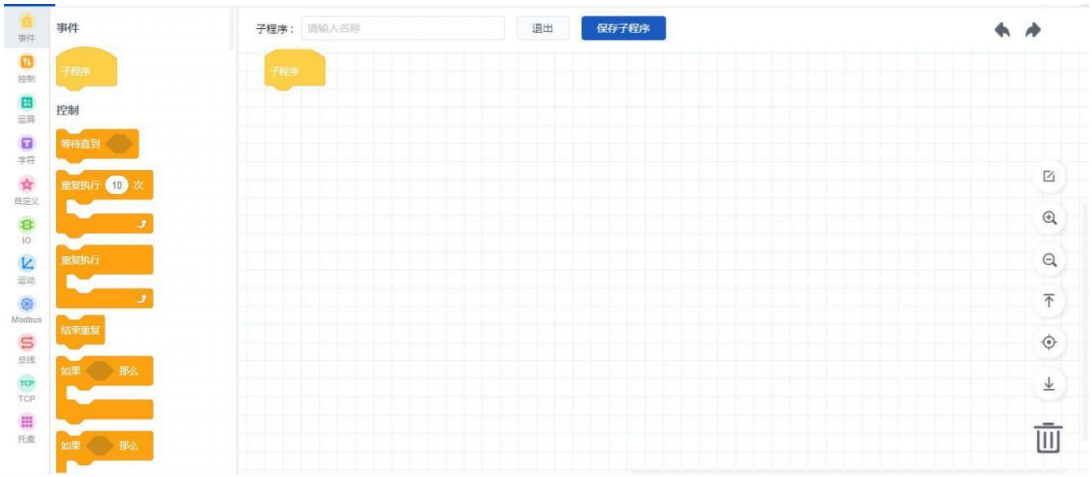
Create a new subroutine

Click to create a new subroutine.

> **illustrate :**
> a subroutine and a function is that a subroutine is not encapsulated; it is essentially a reusable block of code. Calling a subroutine is equivalent to copying and pasting the code from the subroutine into the corresponding location.

Graphical programming supports nested subroutines, which in turn support both graphical and script-based programming, with a maximum nesting level of two. Once a subroutine i s successfully created , the corresponding subroutine block will appear in the block list.

- After selecting block programming, the page changes to the subroutine block programming page, where you can set the subroutine description and write subroutines.



- Selecting script programming will bring up the subroutine script programming window, where you can set the subroutine description and write the subroutine.

subroutine

- Graphical programming subroutine

- Script programming subroutines

Description: Subroutine block. The description is defined by the user when creating the subroutine and is used to call a saved subroutine. Right-click (PC) / long-press (mobile) the block in the block list to modify or delete the subroutine.

I O building block set

The IO block group is used to manage the input and output of the robot's IO terminals. T he value range of the input and output ports is determined by the number of correspond ing terminals on the robot ; please refer to the hardware manual of the corresponding r obot.

Set digital output

Description : Enables or disables

the specified DO. **Parameters :**

- Select the location of the DO terminal, including control cabinets and terminals.
- Select the DO terminal number.
- Choose the output state (on or off).

Determine the digital output status

Description : Determines whether the current

state of the specified DO satisfies the given

condition. **Parameters :**

- Select the location of the DO terminal, including control cabinets and terminals.
- Select the DO terminal number.
- Choose the state that you want to determine as true.

Return value : Returns true if the current state of the specified DO satisfies the condition, oth erwise returns false.

Set a set of digital outputsI O building block set

The IO block group is used to manage the input and output of the robot's IO terminals. T he value range of the input and output ports is determined by the number of correspond ing terminals on the robot ; please refer to the hardware manual of the corresponding r obot.

Set digital output

Description : Enables or disables

the specified DO. **Parameters :**

- Select the location of the DO terminal, including control cabinets and terminals.
- Select the DO terminal number.
- Choose the output state (on or off).

Determine the digital output status

Description : Determines whether the current

state of the specified DO satisfies the given

condition. **Parameters :**

- Select the location of the DO terminal, including control cabinets and terminals.
- Select the DO terminal number.
- Choose the state that you want to determine as true.

Return value : Returns true if the current state of the specified DO satisfies the condition, oth erwise returns false.

Set a set of digital outputs

Description : Set up a group of DOs.

Parameters : After dragging the block into the programming area, click to set DO and status.

- via + or - You can increase or decrease the number of DOs to be set.
- Select the DO terminal number from the drop-down menu on the left.
- Select the desired output state (on or off) from the drop-down menu on the right.

Waiting for a set of numbers to be output

Description : Wait for all the states of a group of

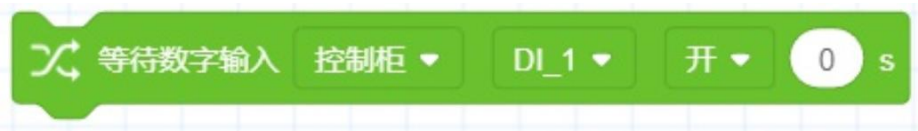
DOs to meet the conditions before continuing

execution. **Parameters :**

- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.
- After dragging the block into the programming area, click to set the DO to wait for.

- By + or - You can increase or decrease the number of DOs that need to be waited for.
- Select the DO terminal number from the drop-down menu.

Waiting for number input



Description: Waits for the specified DI to meet a condition or for a timeout to occur before executing subsequent block instructions. **Parameters :**

- Select the location of the DI terminals, including control cabinets and terminals.
- Select the DI terminal number.
- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.

Waiting for a set of numbers to be entered

Description : Wait for all states in a group of DI processes to meet the conditions before continuing execution. **Parameters :**

- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.
- After dragging the block into the programming area, click to set the DI to wait for.

- via + or - You can increase or decrease the number of DIs to wait for

- Select the DI terminal number from the drop-down menu.

Set analog output

Description: Sets the value of the specified analog output. The meaning of the value (voltage/current) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/ AO pages of the control cabinet .**

parameter :

- Select the number of the analog output terminal.
- The value to be output can be entered directly or other oval blocks that return numerical values can be used.

Get analog output

Description: Retrieves the value of the specified analog output. The meaning of the value (voltage/current) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/ AO pages of the control cabinet .**

Parameter : Select the number of the analog output terminal.

Return value: The voltage or current value currently set by the AO.

Determine the status of the numeric input

Description : Determines whether the

current state of the specified DI satisfies

the given conditions. **Parameters :**

- Select the location of the DI terminals, including control cabinets and terminals.
- Select the DI terminal number.
- Choose the state that you want to determine as true.

Return value : Returns true if the current state of the specified DI satisfies the condition, otherwise returns false.

Get simulated input

Description : Retrieves the value of a specified simulated input.

The meaning of the analog input values (voltage/current) of the control cabinet can be fo und in CBSH Studio . Pro's **monitoring >** View and modify **AI/AO pages on the control cab inet ;**

To obtain the end-point analog input, you first need to set the end-point tool mode to analog input mode by setting the end-point tool mode block.

parameter :

- Select the location of the analog input terminals, including control cabinets and terminals.
- Select the analog input terminal number. **Return value:** Specifies the value of the analog input.

Sports building blocks

The motion block set is used to control the movement of the robotic arm and to perform moti on-related settings.

Point parameters can be added to the project's **point list and then selected here; the bloc ks that control movement also support** dragging out the default variable blocks and repla cing them with other elliptical blocks that return point values.

Move to the target point

Description: Control the robotic arm to move from its current position to a designated point. Drag the block to the programming area, then click to access advanced configuration. **Parameters :**

- Choose the movement mode, supporting joint movement and linear movement.
- Target point. The default point variable block supports the following functions:

- Select a point from the list of points to save.
- Add the current pose of the robotic arm as a new point and select it automatically.
- Use the current pose of the robotic arm to cover the currently selected point.

Advanced configuration

Joint motion has fewer configurable parameters than linear motion. The following figure shows a pop-up window for linear motion as an example. See the explanation below for the differences.

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute Speed: Supported only for linear motion. The absolute speed value is mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): Supported only for linear motion. The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.
- Process I/O settings:

Process I/O settings and stop conditions cannot be selected simultaneously.

Used to set the state of a specified DO when the robot moves to a specified distance or percentage.

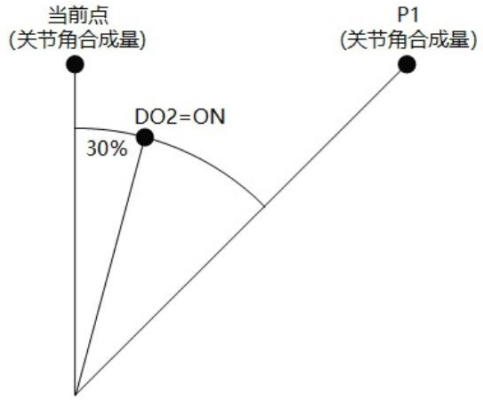
distance value indicates the distance from the starting point, while a negative distance value indicates the distance from the target point. If a smooth transition is set, the robotic arm may not reach the target point, potentially affecting the timing of the DO output.

the movement mode is joint movement, the distance represents the composite angular vector of each joint angle, which is relatively complex to calculate. Therefore, it is recommended to use the percentage mode for a more intuitive result.

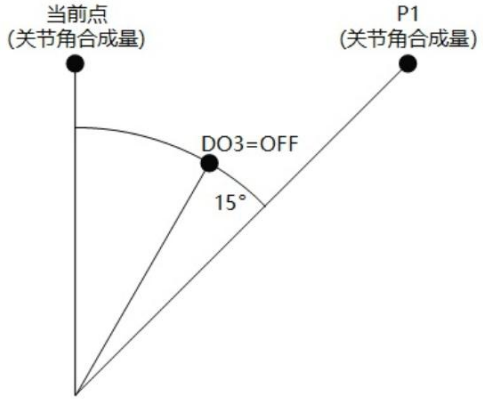
Click below **+** You can add process I/O by clicking on the right. **-** The corresponding process I/O can be deleted.

Below are some examples of process I/O settings.

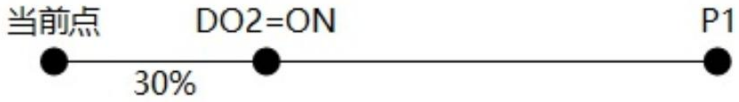
Example 1 : Joint movement to P1, DO index: DO_02, DO state: ON, trigger mode: percentage, distance: 30% means that when the joint moves to a position 30% away from the starting point, DO2 will be set to ON.



Example 2 : Joint movement to P1, DO index: DO_03, DO state: OFF, trigger mode: distance, distance: -15° means that when the joint moves to a position 15° away from the end point, DO3 will be set to OFF.



Example 3 : Moving in a straight line to P1, DO index: DO_02, DO state: ON, trigger mode: percentage, distance: 30% means that when the linear motion reaches a position 30% away from the starting point, DO2 will be set to ON.



Example 4 : Moving in a straight line to P1, DO index: DO_03, DO state: OFF, trigger mode: distance, distance: -15mm means that when the linear motion reaches a position 15mm away from the endpoint, DO3 will be set to OFF.

- Stop condition:

Process I/O settings and stop conditions cannot be selected simultaneously.

Set the stopping conditions for the movement . When the conditions are met, the robot wil l end the current movement and directly execute the next instruction.

Example 1 : Set only one line of condition "when DI1==ON", which means that when DI1 is detected to be ON during the execution of this motion instruction, the current motion is skipped.

Example 2 : Set two conditions "when DI1==ON and DI2~=ON", which means that d uring the execution of this motion instruction, if it is detected that DI1 is ON and DI2 is not ON, the current motion is skipped.

Example 3 : Set two conditions "when DI1==ON, or global variable var<=10", which means that during the execution of this motion instruction , if DI1 is detected to b e ON or global variable var is less than or equal to 10, the current motion is skippe d.

along coordinate system

Description: Controls the robotic arm to offset a specified distance from its current positio n along a specified coordinate system. Drag the block to the programming area and click to access advanced configuration.

parameter :

- Choose the movement mode, supporting relative joint movement and relative linear mov ement.
- Specify the offset of the robotic arm along the user coordinate system or tool coordinate system.
- T h e o f f s e t i n the specified coordinate system, where x, y, z represent

spatial offsets in millimeters; rx, ry, rz represent angular offsets in degrees.

Advanced configuration

Joint motion has fewer configurable parameters than linear motion. The following figure shows a pop-up window for linear motion as an example. See the explanation below for the differences.

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute Speed: Supported only for linear motion. The absolute speed value is mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): Supported only for linear motion. The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: The index of the user coordinate system referenced during offset.
- Tool coordinate system: The index of the tool coordinate system referenced during offset.
- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil
l stop the current movement and directly execute the next instruction.

Get the point after offset along the coordinate system

Description: Offsets a specified point by a specified distance along

a specified coordinate system and returns the new point.

Parameters :

- Select the point to offset.
- Choose which coordinate system the offset is based on for the teaching point; you can ch
oose either the user coordinate system or the tool coordinate system.
- Input the offset distance

in each direction. **Return**
value: The new position after
offset.

Joint displacement movement

Description: Controls the robotic arm joints to move a specified offset from their current
position. Drag the block to the programming area and click to access advanced config
uration .

Parameters : Offset of each joint,

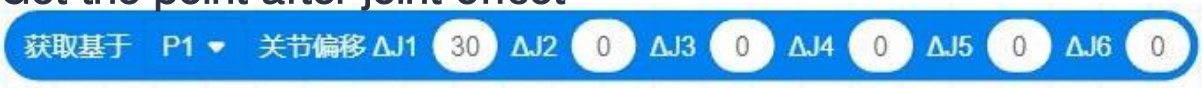
in degrees. **Advanced**

Configuration

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- User coordinate system/tool coordinate system: These two parameters are invalid in this block.

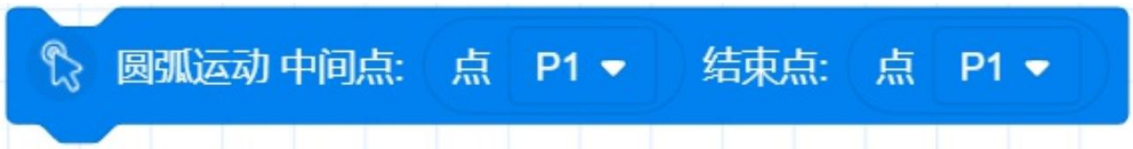
Get the point after joint offset



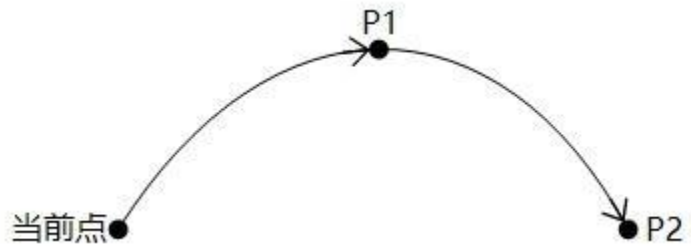
Description: Offsets a specified point along each joint by a specified angle and returns the new offset point. **Parameters :**

- Select the point to offset.
- Input the offset angle for each joint. **Return value:** The new position after offset.

Perform circular motion



Description: Control the robotic arm to move from its current position to a specified point in a Cartesian coordinate system using circular interpolation. The coordinates of the current position must not lie on the straight line defined by the intermediate and ending points. Drag the block to the programming area and click to access advanced configuration.



the movement is calculated by interpolating the postures of the current point and point P2. The posture of point P1 is not included in the calculation (that is, the posture of the robotic arm when it reaches point P1 during the movement may be different from the taught posture).

parameter :

- The midpoint refers to the point used to determine the midpoint of an arc.
- The endpoint is the target point.

Advanced configuration

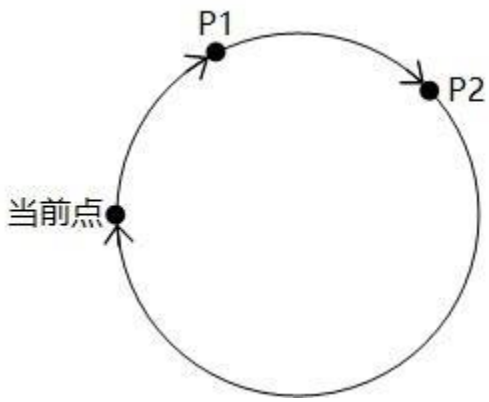
The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute speed (Speed): The absolute speed value of motion, mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.
- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil
l end the current movement and directly execute the next instruction.

Perform circular motion

Description: Control the robotic arm to perform full-circle interpolation motion from its cur
rent position, returning to the current position after a specified number of revolutions. The
coordinates of the current position must not lie on the straight line defined by the midpoin
t and end point. Drag the block to the programming area and click to access advanced co
nfiguration.



the movement is calculated by interpolating the postures of the current point and point P
2. The posture of point P1 is not included in the calculation (that is, the posture of the rob
otic arm when it reaches point P1 during the movement may be different from the taught
posture).

parameter :

- The midpoint refers to point 1 used to define the location of the entire circle.
- The end point refers to point 2, which is used to determine the location of the entire circle.
- Enter the number of full circular motions, ranging from 1 to 999.

Advanced configuration

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute speed (Speed): The absolute speed value of motion, mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.
- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil l stop the current movement and directly execute the next instruction.

Trajectory Reproduction

Description: After the robotic arm moves to the starting point of its trajectory, the trajectory is reproduced. The reproduced trajectory file must be recorded in the trajectory reproduction process. Drag the block to the programming area and click to access advanced configuration.

parameter :

- Choose the motion mode when moving to the starting point of the trajectory, supporting joint motion and linear motion.
- Select the trajectory file you want to reproduce. The currently selected trajectory file will be displayed as a question mark if it is deleted.
- Select the motion speed during reproduction:
  - At a constant speed , the robotic arm will reproduce the trajectory at a uniform global speed.
  - At 0.25x speed , the original speed is scaled proportionally based on the recorded trajectory. In this mode, the robotic arm's movement speed is unaffected by the global speed, and the same applies below.
  - 0.5x speed
  - 1x speed
  - 2x speed

Advanced configuration

The following parameters need to be checked for them to take effect.

- Reproduction interval : The sampling interval of trajectory points, i.e., the sampling

time difference between two adjacent points when generating the trajectory file. Value range:
[8, 1000], unit: ms, default value is 50 (sampling interval when the controller records trajectory files).

- Filtering coefficient : The smaller the value of this parameter, the smoother the reproduced trajectory curve, but the more severe the deformation relative to the original trajectory. Please set an appropriate filtering coefficient according to the smoothness of the original trajectory . Value range: (0,1], 1 indicates that filtering is turned off, and the default value is 0.2.
- User coordinate system: Specifies the user coordinate system index corresponding to the trajectory point. If not specified, the user coordinate system index recorded in the trajectory file is used.
- Tool coordinate system: Specifies the tool coordinate system index corresponding to the trajectory point. If not specified, the tool coordinate system index recorded in the trajectory file is used.

Set the smooth transition ratio

Description: Sets the smoothness ratio during motion. This setting only applies to the current project run. See the General Guidelines for detailed instructions on smooth transitions .

Parameter : Smooth transition ratio, value range: 0~100.

Set joint speed ratio

Description: Sets the speed ratio of joint movements. This setting only applies to the current project run. See the general instructions for detailed speed calculation instructions .

Parameter : Joint velocity ratio, value range: 0~100.

Description: Sets the acceleration ratio for joint movements. This setting only applies to the current project run. For detailed instructions on acceleration calculation, please refer to

设置关节加速度比例

Parameter : Joint acceleration ratio, value range: 0~100.

Set linear velocity ratio

Description: Sets the speed ratio for linear and arc motion. This setting only applies to the current project run. See the general instructions for detailed speed calculation instructions .

Parameter : Ratio of linear and arc speeds, value range: 0~100.

Set the linear acceleration ratio

Description: Sets the acceleration ratio for linear and arc motion. This setting only applies to the current project run. See the general instructions for detailed acceleration calculation instructions .

Parameter : The ratio of linear and arc acceleration, with a value range of 0 to 100.

Set global speed ratio

Description: Sets the global velocity scale for robot movement. This setting only applies to the current project run. See the general instructions for detailed velocity calculation instructions .

Parameter : Global speed ratio, value range: 0~100.

Modify the coordinates of a specified point

Description : Modifies the value of a specified Cartesian coordinate dimension for a specified point. **Parameters :**

- Select the point you want to modify.
- Select the coordinate dimension you want to modify.
- Select the modified value.

Get the coordinates of a specified point

Description : Get the coordinates of a specified point. **Parameters :** Select the point from which you want to obtain the coordinates. **Return value:** The coordinates of the specified point.

Check the feasibility of the exercise

Description: Check the feasibility of the robot moving from the current point to a specified point using a specified motion. The system will calculate the entire motion trajectory and check for any unreachable points along the trajectory .

parameter :

- Choose the movement mode, supporting joint movement and linear movement.
- Select the target point.

Return value:
Check result.

- 0: No errors
- 16: The finish line is near the shoulder, which is unusual.
- 17: The inverse solution at the endpoint has no solution.

- 18: Endpoint Reverse Solution Limit
- 22: Gesture switching error
- 26: The end point is near the wrist, which is strange.
- 27: The finish line is strangely close to the elbow.
- 29: Speed parameter error
- 30: Failed to solve the inverse problem with all parameters
- 32: The trajectory has a shoulder anomaly.
- 33: The trajectory has an inverse solution but no solution point.
- 34: The trajectory has inverse solution limit points.
- 35: The trajectory has a wrist singularity.
- 36: The trajectory has an axial singularity.
- 37: The trajectory contains joint jump points.

Get the value of a specified coordinate dimension of a specified point

Description : Retrieves the value of a

specified Cartesian coordinate dimension

for a given point. **Parameters :**

- Select the point from which you want to obtain the coordinates.
- Select the coordinate dimensions you want to obtain.

Return value: The value of the specified Cartesian coordinate dimension of the specified poin t.

Get the angle of a specified joint at a specified point

Description : Retrieves the angle of

a specified joint at a specified point.

Parameters :

- Select the point from which you want to obtain the joint angle.
- Select the joint from which you want to obtain the angle.

Return value: The angle of the specified joint at the specified point.

joint angle is correctly interpreted as the position.

Description: Given the angles of each joint of a robotic arm, calculate the pose of the robotic arm's end effector in a given Cartesian coordinate system. **Parameters :**

- Select a point; the joint angle at that point will be used in the forward solving calculation.
- User coordinate system index.
- Tool coordinate system index.

Return value: The pose variable obtained from the correct solution, in the format `{pose = {x, y, z, rx, ry, rz} }`

Inversely decompose the pose into joint angles

Description: Given the coordinates of the robotic arm's end effector in a given Cartesian coordinate system, calculate the angles of each joint of the robotic arm. Since Cartesian coordinates only define the spatial coordinates and tilt angles of the TCP (Tilting Coordinate System), the robotic arm can reach the same pose through a variety of different postures. This means that one pose can correspond to multiple joint angles. This block will return the joint angles closest to the current pose of the robotic arm.

parameter :

- Select a point; the pose of that point will be used for inverse kinematics calculations.
- User coordinate system index.
- Tool coordinate system index.

Return value: Returns two variables, which can be printed to view the results. The first variable is the error code: 0 indicates successful reverse engineering, and -1 indicates failure. (No solution); the second variable is the joint angle obtained from the inverse solution, in the format `{joint = {j1, j2, j3, j4, j5, j6} When` the inverse solution fails, all values from `j1` to j6 are 0.

Get the encoder value

Description : Retrieves the current value of the ABZ encoder. **Return value:** The current value of the encoder.

Modbus building blocks

Modbus blocks are used for Modbus communication-related operations. Refer to the corresponding demo for a quick experience of the relevant commands. The Modbus function codes for various registers follow the standard Modbus protocol.

Register type	Read registers	Write a single register	Write multiple registers
Coil Register	0 1	0 5	0 F
Contact register	0 2	-	-
Input register	0 4	-	-
Holding register	0 3	0 6	1 0

Create a Modbus master station

Description: Creating a Modbus network based on the control cabinet's network port. The TCP master station establishes connections with the slave stations. A maximum of 15 slave stations can be connected simultaneously .

parameter :

- Enter the IP address of the Modbus slave station.
- Enter the port number of the Modbus slave station.
- Select the ID of the Modbus slave station.

connecting to the robot's built-in slave station, set the IP address to the robot's IP address (default 192.168.5.1, which can be modified), and set the port to 502 (map1) or 1502 (map2). See Appendix A for details. Modbus register definition .

connecting to a third-party slave station, please refer to the Modbus register address definition description of the corresponding slave station for the range and definition of register address values when reading and writing registers.

Create a Modbus master station based on the terminal RS485

Description: Creating a Modbus based on the end RS485 interface The TCP master station establishes connections with slave stations. It supports a maximum of 15 simultaneous connections .

parameter :

- Enter the IP address of the Modbus slave station.
- Enter the baud rate for the RS485 interface.
- Select the ID of the Modbus slave station.
- Choose whether to include a parity bit.
- Select the stop bit length.

connecting to the robot's built-in slave station, set the IP address to the robot's IP address (default 192.168.5.10, which can be modified), and the port to 60000 (which cannot be set or modified).

connecting to a third-party slave station, please refer to the Modbus register address definition description of the corresponding slave station for the range and definition of register address values when reading and writing registers.

Create a Modbus master station based on RS485

Description: Creating a Modbus based on the control cabinet's RS485 interface The RTU master station establishes connections with slave stations. It supports up to 15 devices connected simultaneously.

parameter :

- Enter the baud rate for the RS485 interface.
- Select the ID of the Modbus slave station.
- Choose whether to include a parity bit.
- Input the length of the data bits; the current version only supports 8 bits.

- Select the stop bit length.

Get the results of creating the main site

Description : Retrieves the result of

creating a Modbus master station.

Return value:

- 0: Modbus master station created successfully
- 1: There are already 15 main sites created; failure to create a new main site.
- 2: Main site initialization failed. We recommend checking the IP address, port, and network connectivity.
- 3: Connection to slave station failed. It is recommended to check whether the slave station was established normally and whether the network is normal.

Waiting for input register

Description: Waits for the value at the specified address in the

input register to meet a condition before continuing execution of

the next instruction. **Parameters :**

- Input the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)
- The conditions that the value at the specified address of the input register must satisfy.

Waiting for holding register

Description: Wait for the value at the specified address in the holding register to meet a certain condition before continuing execution of the next instruction. **Parameters :**

- Hold the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)
- The conditions that must be met to maintain the value at a specified address in a register.

Waiting contact register

Description: Wait for the value at the specified address in the contact register to meet a certain condition before continuing to execute the next instruction. **Parameters :**

- The starting address of the contact register.
- The conditions that the value at the specified address of the contact register must meet.

Waiting coil register

Description: Waits for the value at the specified address in the coil register to meet a condition before continuing execution of the next instruction.

Parameters :

- Coil register start address.
- The conditions that the value at the specified address of the coil register must meet.

Read input register

Description : Retrieves the value at

the specified address of the input

register. **Parameters :**

- Input the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number

(8 bytes, occupying 4 registers) **Return value:** The value at
the address specified in the input register.

Read holding register

Description : Retrieves the value at

the specified address of the

holding register. **Parameters :**

- Hold the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number

(8 bytes, occupying 4 registers) **Return value:** Holds the
value at the specified address of the register.

Read contact register

Description : Retrieves the value at a specified address in the contact register. **Parameter :** The starting address of the contact register.

Return value: The value at the specified address of the contact register.

Read coil register

Description : Retrieves the value at the specified address in the coil register. **Parameter :** The starting address of the coil register.

Return value: The value at the specified address of the coil register.

Continuous reading of coil register

Description : Reads the value at the specified address of the coil register continuously. **Parameters :**

- Coil register start address.
- The number of register bits read sequentially.

Return value: The value at the specified address of the coil register, stored in a table. The first value in the table corresponds to the value at the starting address of the coil register。

Continuous Read Holding Register

Description : Reads the value of the holding register at the specified address continuously.
parameter :

- Hold the starting address of the register.
- The number of values read consecutively.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)

Return value: The value at the specified address of the coil register, stored in a table. The first value in the table corresponds to the value at the starting address of the coil register .

Write to coil register

Description : Writes the specified value to

the specified address in the coil register.

Parameters :

- Coil register start address.
- The value to be written can only be 0 or 1.

Continuous writing to coil register

Description : Writes the specified value

continuously to the specified address in the coil

register.

Parameters :

- Coil register start address.
- Input the value to be written, with multiple values separated by commas, and each digit can only be 0 or 1.

Write to holding register

Description : Writes the specified value to the holding register at the specified address.
parameter :

- Hold the starting address of the register.
- The value you choose to write must correspond to the data type you select.
- Select the data type to write:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)

Shut down the main site

Description : Shut down the Modbus master station and disconnect it from all slave stations.

Bus block group

The bus block group is used to read and write Profinet or Ethernet/IP bus registers. For instructions on using the bus communication function, please refer to the
" CBSH Bus Communication Protocol Document (EtherNet/IP, Profinet)".

> illustrate :
> Magician E6 does not support this set of instructions.

Get the value of the bus register

Description : Retrieves the value

of the specified bus register.

Parameters :

- Select the register type; bus input registers and bus output registers are supported.
- Choose the data type of the register; bool, int, and float are supported.
- Choose the address of the register.

Return value: The value of the specified register. A boolean value can be 0 or 1.

Set the value of the bus register

Description : Sets the value of

the specified bus register.

Parameters :

- Select the register type; currently only bus output registers are supported.
- Choose the data type of the register; bool, int, and float are supported.
- Choose the address of the register.
- Enter the value you want to set. A boolean value can be either 0 or 1.

TCP building block set

The TCP block group is used for TCP communication-related operations. Refer to the corresponding demo for a quick experience of the relevant commands.

Connect SOCKET

Description : Creates a TCP client to communicate

with a specified TCP server. **Parameters :**

- Select the SOCKET number to establish a maximum of 4 TCP communication links.
- The IP address of the TCP server.
- The port of the TCP server.

Get connection SOCKET result

Description : Retrieves TCP

communication connection

results. **Parameter :** Select

SOCKET number.

Return value: 0 for a successful connection, 1 for a failed connection.

Create SOCKET

Description : Creates a TCP server and

waits for clients to connect.

Parameters :

- Select the SOCKET number to establish a maximum of 4 TCP communication links.
- The IP address of the TCP server.
- The port for the TCP server. Do not use the ports listed below that are already in use by th e system, as this will cause server creation to fail.

  7, 13, twenty two, 37,

  139, 445, 502, 503,

  1501, 1502, 1503, 4840, 8172, 9527,

  11740, 22000, 22001, 29999, 30004, 30005, 30006,

  60000~65504, 65506, 65511~65515, 65521, 65522

Get SOCKET creation result

Description : Retrieves the TCP

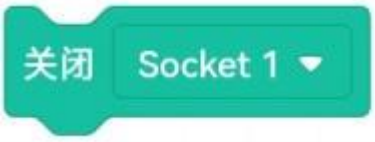
server creation result.

Parameter : Select the

SOCKET number.

Return value: 0 if creation is successful, 1 if creation fails.

Close SOCKET

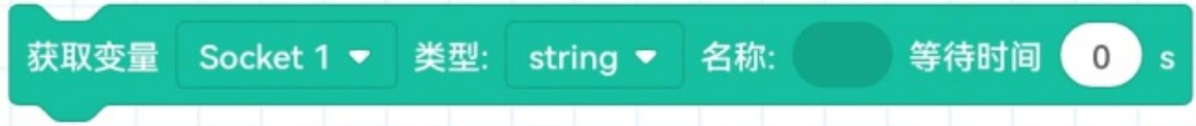


Description : Closes the specified socket,

disconnecting the communication link.

Parameter : Select the socket number.

Reading variables



Description : Reads and saves

variables via TCP communication.

Parameters :

- Select SOCKET number.
- Choose the type of variable to receive; strings and numbers are supported.
- The variable used to store the received data uses the already created variable blocks.
- Enter the timeout period. If set to 0, there will be no timeout, and the system will wait inde finitely until the variable is received.

Get the result of reading the variable

Description : Retrieves the

result of TCP read variables.

Parameter : Select the

SOCKET number.

Return value: Returns 0 if the read is successful, and 1 if the read fails.

Send Variables

Description : Sends variables

via TCP communication.

Parameters :

- Select SOCKET number.
- The data sent can be filled in directly or other oval blocks that return strings or numbers.

Get the result of sending variables

Description : Retrieves the

results of TCP sent variables.

Parameter : Select the

SOCKET number.

Return value: 0 if the transmission is successful, 1 if the transmission fails.

Tray building blocks

A pallet is a container for holding batches of materials according to a set pattern, commo nly used in automated loading and unloading processes. A pallet typically contains an arr ay of recesses, each capable of holding one material. Using pallet commands, a complet e pallet array can be created by teaching a small number of points , and the specific loca tions within the created pallet can be retrieved, enabling rapid automated loading and unl oading by the robot.

Create a tray

Description: Create trays. Supports creating 1D, 2D, and 3D trays. A maximum of 20 tra ys can be created. Creating a tray with the same name will overwrite an existing tray an d will not increase the tray count.

parameter :

on the block , then click the block to bring up the settings

window and configure the pallet parameters. First, select the

pallet dimension; different dimensions of pallets require different

parameters.

- One-dimensional tray

A one -dimensional tray is a set of points equidistantly distributed along a straight line.

- First, fill in the number of objects P1~P2, which is the total number of points on the one-dimensional pallet.
- Then configure points P1 and P2 separately, which can only be selected from the stored point list.

- Two-dimensional tray

A two-dimensional tray is a set of points arrayed on a plane.

- Fill in the number of objects in rows P1~P2 and columns P1~ P4 respectively . The product of the two is the total number of points on the two-dimensional tray .
- Then configure points P1 to P4 respectively, which can only be selected from the stored point list.

- 3D tray

A three-dimensional pallet is a set of points distributed in three dimensions in space, which can be regarded as multiple two-dimensional pallets arranged vertically.

- Enter the number of objects in rows P1~P2, columns P1~P4, and layers P1~P5 respectively. The product of these three numbers is the total number of points in the 3D tray .
- Then configure points P1 to P8 respectively, which can only be selected from the stored point list.

Notice :
If you are using an end-effector tool, make sure that the tool coordinate system corresponding to the end-effector tool is selected when teaching the point.

Get the total number of tray points

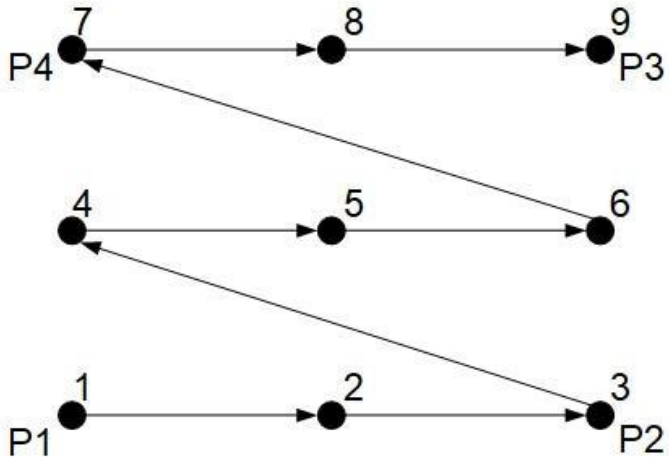
Description : Retrieves the total number of locations for created pallets. **Parameter :** Select the name of the created pallet.

Returns : The total number of points on the specified pallet.

Get tray point

Description : Retrieves the position with a specified index on a specified tray. The index number depends on the order of the positions passed in when the tray was created.

- One -dimensional tray: Point P1 is numbered 1, point P2 is numbered the same as the number of points, and so on.
- Two-dimensional tray: The following diagram uses a 3x3 tray as an example to illustrate the relationship between teaching points and point numbers.



- Three-dimensional tray : Similar to the two-dimensional tray, the index of the first point of the second layer is the index of the last point of the first layer plus one, and so on .

parameter :

- Select the name of the created tray.
- Enter the index of the

point you want to retrieve.
Returns : The coordinates of the point with the corresponding index.

Quick Experience

- Reading and writing Modbus register data
- Data is transmitted via TCP communication.

Reading and writing Modbus register data

Scene Description

To experience how to read and write Modbus data through graphical programming, let's assume we want to implement the following scenario:

The robot creates a Modbus master station, connects to an external slave station, and reads the address of a specified coil register. If the value of that address is 1, the robot moves to point P1.

Programming steps

To achieve the above scenario, the program we need to write is shown in the figure below.

1. Create a master site with the same IP address as the slave site. Keep the default values for port and ID. For quick verification, we've used the robot's built-in slave site, so its IP address is the robot's address (default 192.168.5.1, can be modified).
2. The program checks whether the main site was created successfully. If it was created successfully, the subsequent steps will be executed; otherwise, the program will terminate directly.
3. If the value of the robot coil register 9 has been modified, it may affect the subsequent program logic. Therefore , the value of the coil register 0 needs to be set to 0 first.
4. Wait for the value of coil register 9 to become 1.
5. Control the robot to move to point P1, where P1 is a user-defined save point.
6. Shut down the main site.

If you want to connect to a third-party slave station, please modify the IP address and port in the master station block to the address of the third-party slave station. For the range and definition of register address values when reading and writing registers, please refer to the Modbus register address definition description of the corresponding slave station.

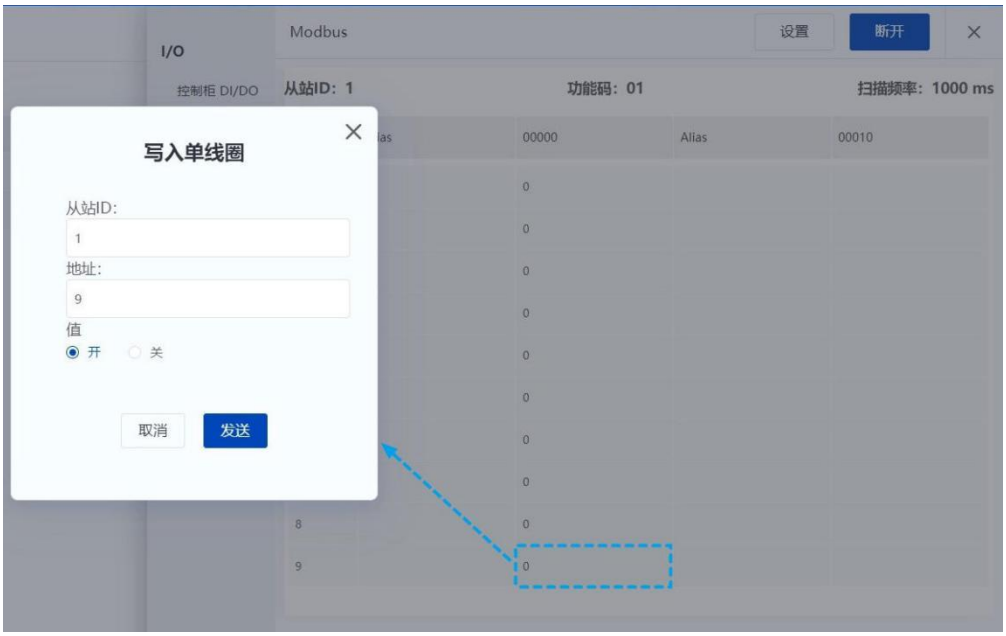
Run the program

If you need to run the program quickly, you can use **the Modbus monitoring** tool to modify the value of the coil register.

1. Open **monitoring > On the Modbus** page, click **the connection in the upper right corner** .
2. The default connection settings are shown in the image below. No modifications are needed; simply click **to connect** .

3. After the Modbus master station is successfully created, double-click (PC) or click (mobile) the cell corresponding to the value of coil register 9 to bring up **the Write Single Coil** window.

4. Change the coil **value** to **"on"** and click **"send"** .



5. Observe whether the robot moves to P1.

Data is transmitted via TCP communication.

Scene Description

To experience how to perform TCP communication through graphical programming, let's assume we want to implement the following scenario:

The robot creates a TCP server, waits for clients to connect and send the "go" command, and then returns "Go". to The message "P1" is sent and the movement begins to point P1.

Programming steps

To achieve the above scenario, the program we need to write is shown in the figure below.

1. Create a TCP server (Socket) 1) The IP address is the robot's address, and the port is customizable.
2. The program checks whether the server creation was successful. If the creation was successful, the subsequent steps will be executed; otherwise, the program will terminate directly.
3. Wait for the client to connect and send a string, then save the received string to the string variable tcp. In `rec v`, string variables need to be created by the user.

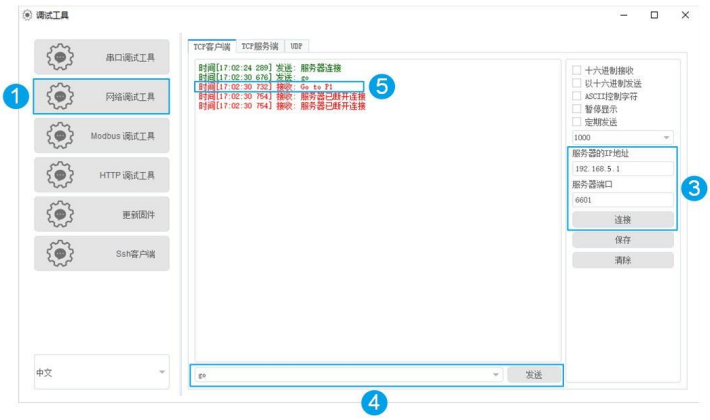
4. Determine if the received string contains "go". If it does, proceed to steps 5 and 6; otherw ise, proceed directly to step 7.
5. Send the string "Go" to P1 is given to the client.
6. Control the robot to move to point P1, where P1 is a user-defined save point.
7. Shut down the TCP server.

Run the program

If you need to run the program quickly, you can use CBSH Studio. The Pro's built-in debuggin g tools act as a TCP client.

1. Open **Settings > Debugging tools** , enter **network debugging tools > TCP client** page.
2. the robot to a point outside of P1 (to facilitate subsequent observation of whether th e robot has executed the movement command), then save and run the program .
3. After the runtime log indicates that the TCP server has been created successfully, modify the server's IP address and port in the debugging tool, and then click Conn ect .
4. the connection is successful, type "go" below the debugging tool and click send.
5. Check if the debugging tool received "Go" to The message

"P1" indicates whether the robot has moved to P1. The image below shows the debugging tool interface, with the numbers corresponding to the steps described above.



Appendix C Script Programming Function Description

Basic Syntax

Basic Concepts

illustrate :
If you wish to systematically learn Lua programming, please search for Lua tutorials online. This manual only lists some basic Lua syntax for your quick reference.

identifier

Identifiers are used to define a variable, function, or other user-defined item. Identifiers b egin with a single letter. A arrive Z or a arrive z or underline _Add zero or more letters, un derscores, and numbers (0 to 1) after the beginning. 9).

It's best to avoid using underscores and uppercase letters as

identifiers, because Lua reserved words also use this convention.

Lua does not allow special characters such as **@** . **$** , and **%** Use

this to define the identifier.

Lua It is a case-sensitive programming language, and the case of all identifiers in the program must be consistent with the examples provided in this manual .

Keywords

The following are listed Lua Reserved keywords. Reserved keywords cannot be used as cons tants, variables, or other user-defined identifiers:

and, break, do, else, elseif, end, false ,for, function, if , in , local, nil, not, or, repeat, return, then, true, until, while, goto

As a general convention, a name beginning with an underscore and followed by a string of uppercase letters (e.g., ...) is used. _____VERSION) is reserved for Lua Inter nal global variables .

Notes

Comments do not affect the execution of the program; they are mainly used to help people w ho read the code understand it.

Single-line comment

In a single line of code, everything following the two hyphens is considered a comment.

Comments can be on a separate line or follow the code.

```
--Single -line comment
print ( "Hello World ! " ) --Single -line comment
```

Multiline comments

Multiline comments begin with "--[[" and end with " The text ends with "--" and the content between the two is considered a comment.

```
-- [[
  Multiline
  comment
  s
  Multiline
  comment
  s
  -- ]]
```

Variables and Data Types

Variables are used to store values, which are then passed as parameters or returned as results. Variables are assigned values using the "=" sign.

In Lua, use local Explicitly declared variables are local variables (effective within the current function). The scope of a local variable extends from its declaration location to the end of the function it belongs to.

Unused local Variables declared explicitly are script-level variables by default (effective in the current script file), and the scope of script-level variables is a single script file.

In addition, you can also use CBSH Studio Pro's **monitoring > The global variables** page sets global variables at the controller level, which can be directly accessed in different script files within the same controller.

Example 1 : This mainly illustrates the difference in scope between local variables and script-level variables.

```
function func () local        -- Define a function that
      a = 1 b = 2             prints a and b when run.
                              -- Local variables
                              -- script-level variables

en d

f unc()   pr                  -- The function is executed, printing the values of a and b as 1 and 2 respectively.
int (a) pr                    -->   nil ( when called outside the variable's scope, it is
int (b)                       printed as nil , the same applies below )
                              --> 2
```

Example 2 : This mainly illustrates that local variables can have the same name as scrip t-level variables, but they only take effect within the function. Code blocks that control fl ow (loops , conditional statements), such as do/end, are also considered functions.

```
a = "a"


for i=10,1,-1 do do
      local a = 6
      print(a)                -- Local variables
end                           --> 6. The variable ' a' is called within the statement block .

print(a)
                          --> a , the script-level variable called outside the statement block .
```

Example 3 : This mainly illustrates the difference between the scope of global variables and the o ther two types of variables, as well as the global retention function of global variables.

```
-- The src0.lua file of Project 1
-- Assuming two global variables , g1 and g2, have
been added to the global variables page...
-- g1 is not globally persistent and has a value of 10.
-- g2 is globally persistent and has a value of 20.

local a = 1 b =
2
print(a)                    --> 1
print(b)                    --> 2

print(g1)                    --> 10
print(g2)                    --> 20
SetGlobalVariable("g1",11)              - -  to global variables that are not gl
                                             obally persistent
SetGlobalVariable ( "g2" , 22 ) p ri    - -  to global variables that are main
nt (g1)                  --> 11              tained globally
p ri nt (g2)                 --> twenty
two

-- The src0.lua file of
Project 2
-- Project 2 runs after Project


1

pr int (a) pr            --> nil
int (b) pr int           --> nil
(g1) pr int              -->   10 ( Non-globally persisted variables were restored to
(g2)                     their values before the modification in Project 1 )
                         --> 22 ( The globally retained variable has been changed to
                         the value modified in Project 1 )
```

Variable names can be any string consisting of letters, underscores, and numbers that do not begin with a number, except for Lua-reserved keywords.

Lua Variables do not require type definitions; simply assigning a value to a variable will a utomatically determine its type based on the value. Assigning a value of a different type t o an already assigned variable within a script will change the variable's type; however, in **monitoring... > The type of** variables set in **the global variable page cannot be changed** , and the type of the assigned value must be consistent with the type selected when the v ariable was created; otherwise, an error will occur when the controller runs the script.

Lua supports various data types, the most common of which include numbers, booleans, strings , and tables. Arrays in Lua are a type of table.

Lua also has a special data type called nil. This indicates an empty string (without any v alid value). For example, printing an unassigned variable will output an empty string. nil value.

number

In Lua, the number is a double-precision floating-point number that supports various operatio ns. The following notations are all treated as number:

- 2
- 2.2
- 0.2
- 2e+1
- 0.2e-1
- 7.8263692594256e-06

Boolean value

boolean The type has only two possible values: true. and false (fake), Lua Bundle false an d nil Seen as False, all others are true, numbers 0 Too true.

String

A string is a sequence of characters consisting of numbers, letters, and underscores. A string can be represented in the following three ways:

- A string of characters between single quotes.
- A string of characters between double quotes.
- [[and]] A string of characters.

When performing arithmetic operations on a numeric string, Lua... It will attempt to convert th is numeric string into a single number.

```
-- Use single quotes to define strings.
local str1 = Hello, Lua!'          print (str1)
- - Output : Hello, Lua!

-- Use double quotes to define strings.
local str2 = Hello, Lua!           print (str2)
- - Output : Hello, Lua!

-- use [[ and ]]       local定义多行字符串
str3 = [[        This is a
multi-line string in Lua.
]]
pr int (str3)
-- Output :
-- This is a
-- string in          multi-line
                      Lua.
--  Lua Automatically convert numeric strings to numbers and perform
arithmetic operations.                  local numStr = "10"
local result = numStr + 20 -- Automatically "10" Convert to number 10
print (result)  - - Output :  30
```

surface

A tab is a set of indexed data.

- The simplest way to construct a table is by using {}, which creates an empty table. Alterna tively, you can directly initialize the table.
- The table is actually an associative array, which can use values of any type as indices, but these values cannot be of any type. nil.
- The table has a variable size and can be expanded as needed.
- The length of a table can be obtained using the # symbol.

```
-- Initialize a table with a contiguous index.
local tbl = {[1] = 2, [2] = 6, [3] = 34, [4] = 5}
print ( "tbl") length " , #tbl) --Output : 4
```

In this example, `tbl` Index `[1]` [`4]` It is continuous, therefore `#tbl` Correctly returned length `4` .

array

An array is a collection of elements of the same data type arranged in a certain order; it can be a one-dimensional array or a multi-dimensional array.

In Lua, arrays are of type `table`, and index keys can be represented by integers. The size of an array is not fixed.

- One-dimensional array: The simplest array, whose logical structure is a linear list.
- Multidimensional arrays: These are arrays that contain other arrays or one-dimensional arrays whose index keys correspond to an array. In Lua, array indices default to starting from... It can start from 1, but you can also specify from. 0 Or it can start with a negative number. Accessing an array element using a non-existent index will return nil.

Example 1 : A one-dimensional array can be used to loop through its elements using a for lo op.

```
-- Create a one-dimensional array
local array = { "Lua" , "Tutorial "

-- Use a for loop to iterate through the array, starting from index 1.
for i = 1 , #array do
    p rint ( array[i]) --Output : Lua Tutorial
en d
```

Example 2 : A one-dimensional array containing numeric elements

```
-- Create a one-dimensional array whose internal elements are numbers.
local numbers = { 10 , 20 , 30 , 40 , 50 }

-- Use a for loop to iterate through the array, starting from index 1.
for i = 1 , #numbers do p rint
    (nu mbers[i]) - - Output : 10 20 30 40 50
end
```

Example 3 : Arrays with custom indices

```
-- Create an array and use negative and positive numbers as indices.
 local array = {} for i =
                    -2 , 2 do
                array[ i] = i  *  2 + 1 -- 为数组赋值
en d

-- Use a for loop to iterate through the -2 到 2 的索引...
for i = -2 , 2 do              print
       (array[i])    --                    输出: -3 -1 1 3 5
en d
```

Example 4 : A 3x3 array of multidimensional arrays

```
-- Initialize array
array = {}
for i=1,3 do
  array[i] = {}
       for j=1,3 do
          array[i][j] = i*j
       end
end

-- 访问数组
for i=1,3 do
    for j = 1 , 3 do                pr
         int (array[i][j])                    --打印结果分别为: 1 2 3 2 4 6 3 6 9
    en d
en d
```

Operators

Arithmetic operators

Instruction symbols	illustrate
+	addition operation
-	Subtraction
*	Multiplication operations
/	Floating-point division operation
//	Floor division
%	Remainder division operation
^	Exponentiation

Example :

```
a = 20
b = 5
prin t (a+b)                --打印a加b的结果：25
prin t (ab) prin            --打印a减b的结果：15
t (a*b) prin t              --打印a乘b的结果：100
(a/b) pr int                --打印a除以b的结果：4
(a//b) prin t                --打印a整除b的结果：4
(a%b) pr int                --打印a除以b的余数结果：0
(a^b)                       --打印a的b次幂的结果：3200000
```

Bitwise operators

Instruction symbols	illustrate
&	Bitwise AND operation
\	Bitwise OR operation
~	Bitwise XOR operation
<<	Bitwise left shift operation
> >	Bitwise right shift operation

Example :

```
pr int (a&b)                --打印a和b按位与的结果：4
```

```
prin t (a|b)                Print the result of the bitwise
prin t (a~b) pr             OR operation between a and b :
int (a<<b) pr               21
int (a>>b)                  --Print the result of XORing a
                            with b : 17
                              --Print the result of shifting 'a' to the left by ' b ' units: 640
                              --Print the result of shifting 'a' to the right by ' b ' units: 0
```

Relational operators

Instruction symbols	illustrate
= =	equal
~ =	Not equal to
< =	Less than or equal to
> =	Greater than or equal to
<	Less than
>	Greater than

Example :

```
a = 20                      --Create variable
b = 5                           a
pr int (a==b) pr                --Create
int (a~=b) pr                   variable b
int (a<=b) pr                   --Print the comparison result of a equal
int (a>=b) prin                 to b : false
t (a<b) pr int                  --Print the comparison result of a not
(a>b)                           equal to b : true
                                --Print the comparison results of a being
                                less than or equal to b : f alse
                                --Print the comparison result of a being
                                greater than or equal to b : true
                            --Print the comparison result of a
                            being less than b : false
                            --Print the comparison result of a
                            being greater than b : true
```

Logical operators

Instruction symbols	illustrate
a nd	The logical AND operator . The result is true only if both sides are true; if either side is false, the result is false.
or	The logical OR operator, when either side returns true, The result is true; only if both sides are false, the result is false.
n ot	The logical NOT operator directly negates the result of the comparison.

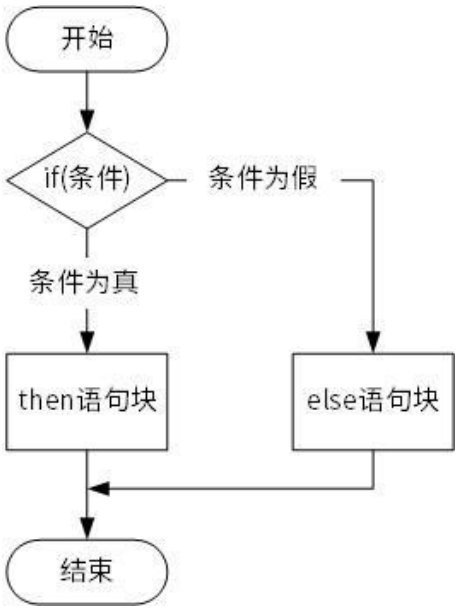
```
a=true
b=false
print(a and b)          --Output :  The function returns true only if both a and b are true .
print(a or b)           -- Output :  true , as long as either a or b is true. It returns
                        true.
print(not a)            --Output :  `false` means `a` is true , and its negation is false.
```

```
print(not b)            -- Output :  If b is true , b is false , and the
print(not (20>5))       negation is true.
                        --Output :  False , 20>5 is true , and the negation
                        is false.
```

Instruction symbols	illustrate
if… then… els e…end	`if` conditional statement evaluates each condition sequentially from t op to bottom. If a condition is true , the corresponding code block is e xecuted, and subsequent condition checks are ignored.
wh ile… d o …end	The while loop control instruction. When the condition is... true This allo ws the program to repeatedly execute certain statements. Before exec uting a statement, it first checks if the condition is true. true
for r…do… e nd	The `for` loop directive repeatedly executes a specified statement, with th e number of repetitions allowed. for Control in statement
re peat… u n til()	The `repeat` loop control instruction. It repeats the loop execution until... U ntil the specified condition is true.

if conditional statement

The parentheses following the `if` statement contain a conditional expression, the result of which can be any value in Lua. Bundle false and nil Consider it false, and everything else true; numbers. 0 It is also true. When the expression is true, the then statement block is executed; when the expression is false, if there is an else statement block, the else statement block is executed ; otherwise, the statements after end are executed directly.

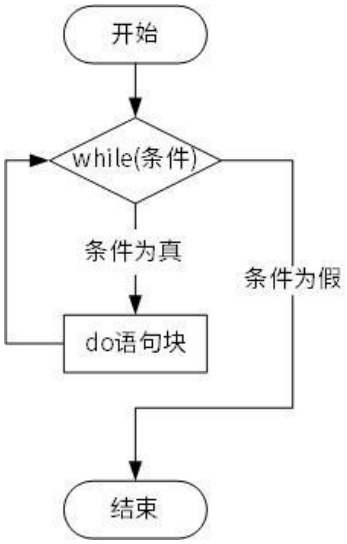


If statements can be nested; below is a typical example.

```
a = 100 ; b
= 200 ;
--[ Check conditions] --]
if (a == 100 ) th
en
      -- [The following if condition is executed when the if condition is true-- ]
      if (b == 200 ) th
      en
            -- [ This block of statements is executed when the if condition is true-- ]
            prin t ( "a The value is : " , a ) -- a The value is :
            100 prin t ( "b The value is : " , b ) -- The value of
            b is : 200
      en d
el se
      -- [ The following block of statements is executed
      when the first if condition is false-- ] pr int ( "a is not
      equal to 100" )
en d
```

while cyclic control command

following the while loop contain the conditional expression. If the expression is true, the do statement block is executed, and then the condition is re-evaluated; if the expression is false, the statement after end is executed directly.



Example :

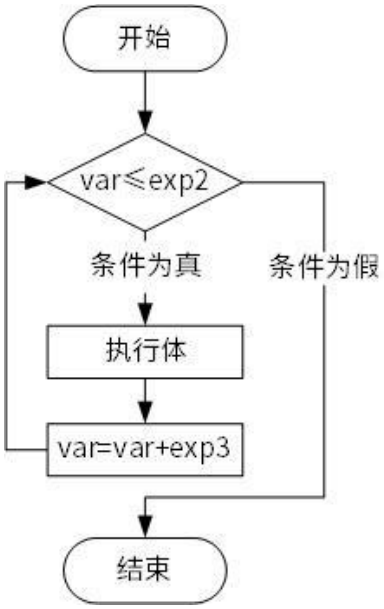
```
a = 10
while ( a < 20 do
      print ( "a The value is : " , a) -- Execute 10 times, output values
      from 10 to 19. a = a+ 1
en d
```

for loop control instructions

The syntax of the for loop is as follows:

```
for  var=exp1,exp2,exp3  do
        <executor>
en d
```

the variable var is exp1. After each execution of the <executive body>, var is incremente d by exp3 (the value of exp3 can be negative or not specified , and the default value is 1) until var is greater than exp2.

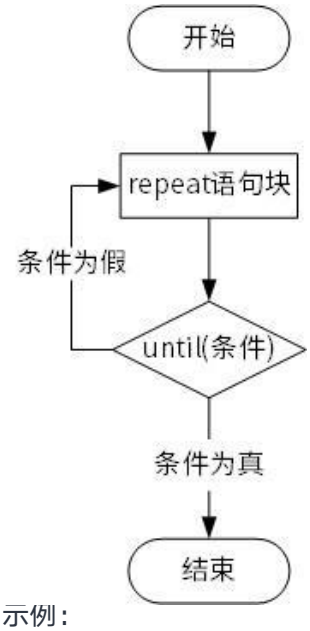


Example :

```
for i = 10 , 1 , -1 do
        print (i) --          执行10次，输出值为10到1
en d
```

re peat loop control command

The repeat loop is similar to the while loop, but the main difference is that while the cond ition is checked before the statement to be looped is executed, and the loop continues if the condition is true ; while repeat the condition is checked after the statement to be lo oped is executed, and the loop continues if the condition is false.



示例：

```
a = 10
repeat
    print("a的值为:", a) -- a = a + 1   Execute 5 times, output value is 10 to 15
until(a > 15)
```

Description : Set up a group of DOs.

Parameters : After dragging the block into the programming area, click to set DO and status.

- via + or - You can increase or decrease the number of DOs to be set.
- Select the DO terminal number from the drop-down menu on the left.
- Select the desired output state (on or off) from the drop-down menu on the right.

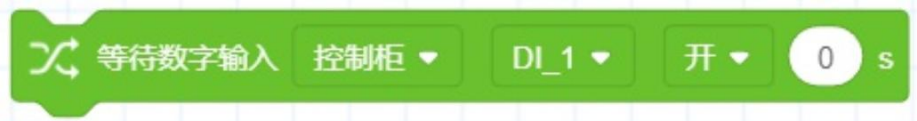
Waiting for a set of numbers to be output

Description : Wait for all the states of a group of DOs to meet the conditions before continuing execution. **Parameters :**

- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.
- After dragging the block into the programming area, click to set the DO to wait for.

- By + or - You can increase or decrease the number of DOs that need to be waited for.
- Select the DO terminal number from the drop-down menu.

Waiting for number input



Description: Waits for the specified DI to meet a condition or

for a timeout to occur before executing subsequent block

instructions. **Parameters :**

- Select the location of the DI terminals, including control cabinets and terminals.
- Select the DI terminal number.
- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.

Waiting for a set of numbers to be entered

Description : Wait for all states in a group of DI

processes to meet the conditions before

continuing execution. **Parameters :**

- Choose the state you want to wait for (on or off).
- The waiting timeout period, if set to 0, will wait until the condition is met.
- After dragging the block into the programming area, click to set the DI to wait for.

- via **+** or **-** You can increase or decrease the number of DIs to wait for.
- Select the DI terminal number from the drop-down menu.

Set analog output

Description: Sets the value of the specified analog output. The meaning of the value (voltage/current) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/ AO pages of the control cabinet .**

parameter :

- Select the number of the analog output terminal.
- The value to be output can be entered directly or other oval blocks that return numerical v alues can be used.

Get analog output

Description: Retrieves the value of the specified analog output. The meaning of the value (voltage/current) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/ AO pages of the control cabinet .**

Parameter : Select the number of the analog output terminal.

Return value: The voltage or current value currently set by the AO.

Determine the status of the numeric input

Description : Determines whether the

current state of the specified DI satisfies

the given conditions. **Parameters :**

- Select the location of the DI terminals, including control cabinets and terminals.
- Select the DI terminal number.
- Choose the state that you want to determine as true.

Return value : Returns true if the current state of the specified DI satisfies the condition, other wise returns false.

Get simulated input

Description : Retrieves the value of a specified simulated input.

The meaning of the analog input values (voltage/current) of the control cabinet can be fo und in CBSH Studio . Pro's **monitoring >** View and modify **AI/AO pages on the control cab inet ;**

To obtain the end-point analog input, you first need to set the end-point tool mode to analog input mode by setting the end-point tool mode block.

parameter :

- Select the location of the analog input terminals, including control cabinets and terminals.
- Select the analog input terminal number. **Return value:** Specifies the value of the analog input.

Sports building blocks

The motion block set is used to control the movement of the robotic arm and to perform moti on-related settings.

Point parameters can be added to the project's **point list and then selected here; the bloc ks that control movement also support** dragging out the default variable blocks and repla cing them with other elliptical blocks that return point values.

Move to the target point

Description: Control the robotic arm to move from its current position to a designated point. Drag the block to the programming area, then click to access advanced configuration. **Parameters :**

- Choose the movement mode, supporting joint movement and linear movement.
- Target point. The default point variable block supports the following functions:

- Select a point from the list of points to save.
- Add the current pose of the robotic arm as a new point and select it automatically.
- Use the current pose of the robotic arm to cover the currently selected point.

Advanced configuration

Joint motion has fewer configurable parameters than linear motion. The following figure shows a pop-up window for linear motion as an example. See the explanation below for the differences.

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.

- Absolute Speed: Supported only for linear motion. The absolute speed value is mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): Supported only for linear motion. The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.
- Process I/O settings:

Process I/O settings and stop conditions cannot be selected simultaneously.

Used to set the state of a specified DO when the robot moves to a specified distance or percentage.

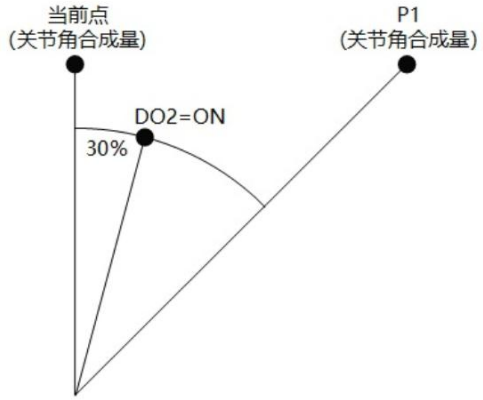
distance value indicates the distance from the starting point, while a negative distance value indicates the distance from the target point. If a smooth transition is set, the robotic arm may not reach the target point, potentially affecting the timing of the DO output.

the movement mode is joint movement, the distance represents the composite angular vector of each joint angle, which is relatively complex to calculate. Therefore, it is recommended to use the percentage mode for a more intuitive result.

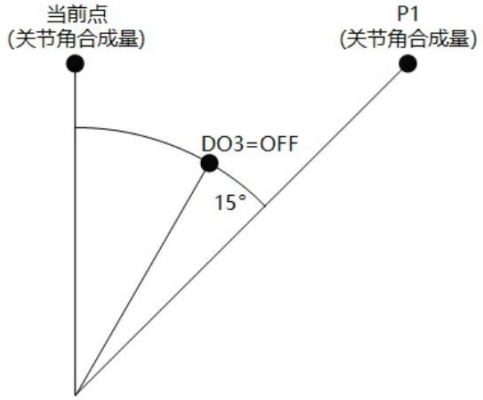
Click below **+** You can add process I/O by clicking on the right. **-** The corresponding process I/O can be deleted.

Below are some examples of process I/O settings.

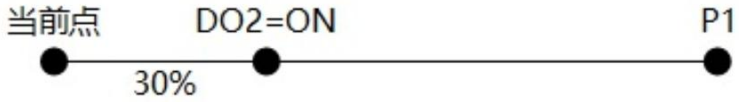
Example 1 : Joint movement to P1, DO index: DO_02, DO state: ON, trigger mode: percentage, distance: 30% means that when the joint moves to a position 30% away from the starting point, DO2 will be set to ON.



Example 2 : Joint movement to P1, DO index: DO_03, DO state: OFF, trigger mode: distance, distance: -15° means that when the joint moves to a position 15° away from the end point, DO3 will be set to OFF.



Example 3 : Moving in a straight line to P1, DO index: DO_02, DO state: ON, trigger mode: percentage, distance: 30% means that when the linear motion reaches a position 30% away from the starting point, DO2 will be set to ON.



Example 4 : Moving in a straight line to P1, DO index: DO_03, DO state: OFF, trigger mode: distance, distance: -15mm means that when the linear motion reaches a position 15mm away from the endpoint, DO3 will be set to OFF.

- Stop condition:

Process I/O settings and stop conditions cannot be selected simultaneously.

Set the stopping conditions for the movement . When the conditions are met, the robot wil l end the current movement and directly execute the next instruction.

Example 1 : Set only one line of condition "when DI1==ON", which means that when DI1 is detected to be ON during the execution of this motion instruction, the current motion is skipped.

Example 2 : Set two conditions "when DI1==ON and DI2~=ON", which means that d uring the execution of this motion instruction, if it is detected that DI1 is ON and DI2 is not ON, the current motion is skipped.

Example 3 : Set two conditions "when DI1==ON, or global variable var<=10", which means that during the execution of this motion instruction , if DI1 is detected to b e ON or global variable var is less than or equal to 10, the current motion is skippd.

along coordinate system

Description: Controls the robotic arm to offset a specified distance from its current positio n along a specified coordinate system. Drag the block to the programming area and click to access advanced configuration.

parameter :

- Choose the movement mode, supporting relative joint movement and relative linear mov ement.
- Specify the offset of the robotic arm along the user coordinate system or tool coordinate system.
- T h e o f f s e t i n the specified coordinate system, where x, y, z represent spatial offsets in millimeters; rx, ry, rz represent angular offsets in degrees.

Advanced configuration

Joint motion has fewer configurable parameters than linear motion. The following figure shows a pop-up window for linear motion as an example. See the explanation below for the differences.

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute Speed: Supported only for linear motion. The absolute speed value is mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): Supported only for linear motion. The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: The index of the user coordinate system referenced during offset.
- Tool coordinate system: The index of the tool coordinate system referenced during offset.

- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil l stop the current movement and directly execute the next instruction.

Get the point after offset along the coordinate system

Description: Offsets a specified point by a specified distance along

a specified coordinate system and returns the new point.

Parameters :

- Select the point to offset.
- Choose which coordinate system the offset is based on for the teaching point; you can ch oose either the user coordinate system or the tool coordinate system.
- Input the offset distance

in each direction. **Return value:** The new position after offset.

Joint displacement movement

Description: Controls the robotic arm joints to move a specified offset from their current position. Drag the block to the programming area and click to access advanced config uration .

Parameters : Offset of each joint,

in degrees.

Advanced Configuration

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

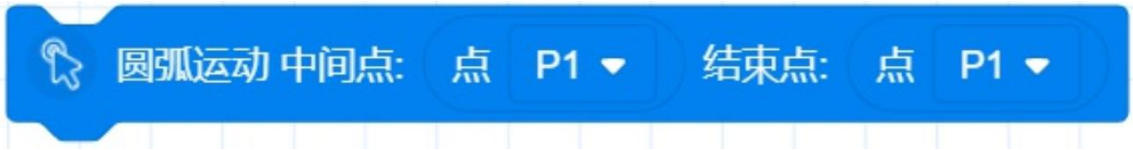
- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- User coordinate system/tool coordinate system: These two parameters are invalid in this block.

Get the point after joint offset

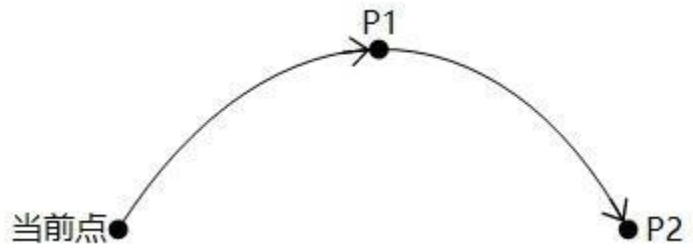
Description: Offsets a specified point along each joint by a specified angle and returns the new offset point. **Parameters :**

- Select the point to offset.
- Input the offset angle for each joint. **Return value:** The new position after offset.

Perform circular motion



Description: Control the robotic arm to move from its current position to a specified point in a Cartesian coordinate system using circular interpolation. The coordinates of the current position must not lie on the straight line defined by the intermediate and ending points. Drag the block to the programming area and click to access advanced configuration.



the movement is calculated by interpolating the postures of the current point and point P2. The posture of point P1 is not included in the calculation (that is, the posture of the robotic arm when it reaches point P1 during the movement may be different from the taught posture).

parameter :

- The midpoint refers to the point used to determine the midpoint of an arc.
- The endpoint is the target point.

Advanced configuration

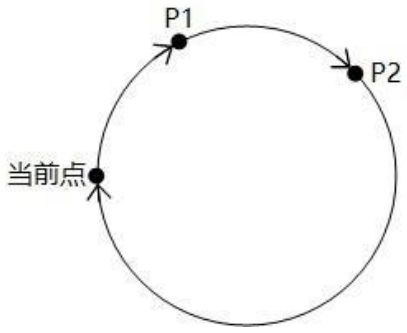
The following parameters will only take effect after being checked. For detailed explanatios of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute speed (Speed): The absolute speed value of motion, mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.
- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil l end the current movement and directly execute the next instruction.

Perform circular motion

Description: Control the robotic arm to perform full-circle interpolation motion from its cur rent position, returning to the current position after a specified number of revolutions. The coordinates of the current position must not lie on the straight line defined by the midpoin t and end point. Drag the block to the programming area and click to access advanced co nfiguration.



the movement is calculated by interpolating the postures of the current point and point P 2. The posture of point P1 is not included in the calculation (that is, the posture of the rob otic arm when it reaches point P1 during the movement may be different from the taught posture).

parameter :

- The midpoint refers to point 1 used to define the location of the entire circle.
- The end point refers to point 2, which is used to determine the location of the entire circle.
- Enter the number of full circular motions, ranging from 1 to 999.

Advanced configuration

The following parameters will only take effect after being checked. For detailed explanations of the parameters, please refer to the general instructions .

- Speed ratio (V): The ratio of motion speed, with a value range of 1 to 100.
- Absolute speed (Speed): The absolute speed value of motion, mutually exclusive with V.
- Acceleration: The proportion of motion acceleration, with a value range of 1 to 100.
- Smooth transition (CP): Smooth transition ratio, value range: 0~100.
- Transition radius (R): The radius of the transition curve, mutually exclusive with CP.
- User coordinate system: User coordinate system index for point parameters.
- Tool coordinate system: Tool coordinate system index for point position parameters.

- Stop condition:

Set the stopping conditions for the movement . When the conditions are met, the robot wil l stop the current movement and directly execute the next instruction.

Trajectory Reproduction

Description: After the robotic arm moves to the starting point of its trajectory, the trajector y is reproduced. The reproduced trajectory file must be recorded in the trajectory reprodu ction process. Drag the block to the programming area and click to access advanced con figuration.

parameter :

- Choose the motion mode when moving to the starting point of the trajectory, supporting j oint motion and linear motion.
- Select the trajectory file you want to reproduce. The currently selected trajectory file will b e displayed as a question mark if it is deleted.
- Select the motion speed during reproduction:
  - At a constant speed , the robotic arm will reproduce the trajectory at a uniform glob al speed.
  - At 0.25x speed , the original speed is scaled proportionally based on the recorde d trajectory. In this mode, the robotic arm's movement speed is unaffected by th e global speed, and the same applies below.
  - 0.5x speed
  - 1x speed
  - 2x speed

Advanced configuration

The following parameters need to be checked for them to take effect.

- Reproduction interval : The sampling interval of trajectory points, i.e., the sampling time difference between two adjacent points when generating the trajectory file. Value range:
  [8, 1000], unit: ms, default value is 50 (sampling interval when the controller records trajectory files).
- Filtering coefficient : The smaller the value of this parameter, the smoother the reproduced trajectory curve, but the more severe the deformation relative to the original trajectory. Please set an appropriate filtering coefficient according to the smoothness of the original trajectory . Value range: (0,1], 1 indicates that filtering is turned off, and the default value is 0.2.
- User coordinate system: Specifies the user coordinate system index corresponding to the trajectory point. If not specified, the user coordinate system index recorded in the trajectory file is used.
- Tool coordinate system: Specifies the tool coordinate system index corresponding to the trajectory point. If not specified, the tool coordinate system index recorded in the trajectory file is used.

Set the smooth transition ratio

Description: Sets the smoothness ratio during motion. This setting only applies to the current project run. See the General Guidelines for detailed instructions on smooth transitions.

Parameter : Smooth transition ratio, value range: 0~100.

Set joint speed ratio

Description: Sets the speed ratio of joint movements. This setting only applies to the curre nt project run. See the general instructions for detailed speed calculation instructions .

Parameter : Joint velocity ratio, value range: 0~100.

Description: Sets the acceleration ratio for joint movements. This setting only applies to th e current project run. For detailed instructions on acceleration calculation, please refer to the General Specifications .

Parameter : Joint acceleration ratio, value range: 0~100.

Set linear velocity ratio

Description: Sets the speed ratio for linear and arc motion. This setting only applies to the current project run. See the general instructions for detailed speed calculation instructions .

Parameter : Ratio of linear and arc speeds, value range: 0~100.

Set the linear acceleration ratio

Description: Sets the acceleration ratio for linear and arc motion. This setting only applies to the current project run. See the general instructions for detailed acceleration calculatio n instructions .

Parameter : The ratio of linear and arc acceleration, with a value range of 0 to 100.

设置关节加速度比例

Set global speed ratio

Description: Sets the global velocity scale for robot movement. This setting only applies to the current project run. See the general instructions for detailed velocity calculation instructions .

Parameter : Global speed ratio, value range: 0~100.

Modify the coordinates of a specified point

Description : Modifies the value of a specified Cartesian coordinate dimension for a specified point. **Parameters :**

- Select the point you want to modify.
- Select the coordinate dimension you want to modify.
- Select the modified value.

Get the coordinates of a specified point

Description : Get the coordinates of a specified point. **Parameters :** Select the point from which you want to obtain the coordinates.

Return value: The

coordinates of the specified

point.

Check the feasibility of the exercise

Description: Check the feasibility of the robot moving from the current point to a specified point using a specified motion. The system will calculate the entire motion trajectory and check for any unreachable points along the trajectory .

parameter :

- Choose the movement mode, supporting joint movement and linear movement.
- Select the

target point.

Return value:

Check result.

- 0: No errors
- 16: The finish line is near the shoulder, which is unusual.
- 17: The inverse solution at the endpoint has no solution.
- 18: Endpoint Reverse Solution Limit
- 22: Gesture switching error
- 26: The end point is near the wrist, which is strange.
- 27: The finish line is strangely close to the elbow.
- 29: Speed parameter error
- 30: Failed to solve the inverse problem with all parameters
- 32: The trajectory has a shoulder anomaly.
- 33: The trajectory has an inverse solution but no solution point.
- 34: The trajectory has inverse solution limit points.
- 35: The trajectory has a wrist singularity.
- 36: The trajectory has an axial singularity.
- 37: The trajectory contains joint jump points.

Get the value of a specified coordinate dimension of a specified point

Description : Retrieves the value of a

specified Cartesian coordinate dimension

for a given point. **Parameters :**

- Select the point from which you want to obtain the coordinates.
- Select the coordinate dimensions you want to obtain.

Return value: The value of the specified Cartesian coordinate dimension of the specified poin t.

Get the angle of a specified joint at a specified point

Description : Retrieves the angle of

a specified joint at a specified point.

Parameters :

- Select the point from which you want to obtain the joint angle.
- Select the joint from which you want to obtain the angle.

Return value: The angle of the specified joint at the specified point.

joint angle is correctly interpreted as the position.

Description: Given the angles of each joint of a robotic arm, calculate the

pose of the robotic arm's end effector in a given Cartesian coordinate

system. **Parameters :**

- Select a point; the joint angle at that point will be used in the forward solving calculation.
- User coordinate system index.
- Tool coordinate system index.

Return value: The pose variable obtained from the correct solution, in the format `{pose = {x, y, z, r x, ry, rz} }`

Inversely decompose the pose into joint angles

Description: Given the coordinates of the robotic arm's end effector in a given Cartesian coordinate system, calculate the angles of each joint of the robotic arm. Since Cartesian coordinates only define the spatial coordinates and tilt angles of the TCP (Tilting Coordinate System), the robotic arm can reach the same pose through a variety of different postures. This means that one pose can correspond to multiple joint angles. This block will return the joint angles closest to the current pose of the robotic arm.

parameter :

- Select a point; the pose of that point will be used for inverse kinematics calculations.
- User coordinate system index.
- Tool coordinate system index.

Return value: Returns two variables, which can be printed to view the results. The first variable is the error code: 0 indicates successful reverse engineering, and -1 indicates failure. (No solution); the second variable is the joint angle obtained from the inverse solution, in the format `{joint = {j1, j2, j3, j4, j5, j6} When` the inverse solution fails, all values from `j1` to j6 are 0.

Get the encoder value

Description : Retrieves the current value of the ABZ encoder. **Return value:** The current value of the encoder.

Modbus building blocks

Modbus blocks are used for Modbus communication-related operations. Refer to the corresponding demo for a quick experience of the relevant commands. The Modbus function codes for various registers follow the standard Modbus protocol.

Register type	Read registers	Write a single register	Write multiple registers
Coil Register	0 1	0 5	0 F
Contact register	0 2	-	-
Input register	0 4	-	-
Holding register	0 3	0 6	1 0

Create a Modbus master station

Description: Creating a Modbus network based on the control cabinet's network port. The TCP master station establishes connections with the slave stations. A maximum of 15 slave stations can be connected simultaneously .

parameter :

- Enter the IP address of the Modbus slave station.
- Enter the port number of the Modbus slave station.
- Select the ID of the Modbus slave station.

connecting to the robot's built-in slave station, set the IP address to the robot's IP address (default 192.168.5.1, which can be modified), and set the port to 502 (map1) or 1502 (map2). See Appendix A for details. Modbus register definition .

connecting to a third-party slave station, please refer to the Modbus register address definition description of the corresponding slave station for the range and definition of register address values when reading and writing registers.

Create a Modbus master station based on the terminal RS485

Description: Creating a Modbus based on the end RS485 interface The TCP master

station establishes connections with slave stations. It supports a maximum of 15 simultaneous connections .

parameter :

- Enter the IP address of the Modbus slave station.
- Enter the baud rate for the RS485 interface.
- Select the ID of the Modbus slave station.
- Choose whether to include a parity bit.
- Select the stop bit length.

connecting to the robot's built-in slave station, set the IP address to the robot's IP address (default 192.168.5.10, which can be modified), and the port to 60000 (which cannot be set or modified).

connecting to a third-party slave station, please refer to the Modbus register address definition description of the corresponding slave station for the range and definition of register address values when reading and writing registers.

Create a Modbus master station based on RS485

Description: Creating a Modbus based on the control cabinet's RS485 interface The RTU master station establishes connections with slave stations. It supports up to 15 devices connected simultaneously.

parameter :

- Enter the baud rate for the RS485 interface.
- Select the ID of the Modbus slave station.
- Choose whether to include a parity bit.
- Input the length of the data bits; the current version only supports 8 bits.
- Select the stop bit length.

Get the results of creating the main site

Description : Retrieves the result of

creating a Modbus master station.

Return value:

- 0: Modbus master station created successfully
- 1: There are already 15 main sites created; failure to create a new main site.
- 2: Main site initialization failed. We recommend checking the IP address, port, and network connectivity.
- 3: Connection to slave station failed. It is recommended to check whether the slave station was established normally and whether the network is normal.

Waiting for input register

Description: Waits for the value at the specified address in the input register to meet a condition before continuing execution of the next instruction. **Parameters :**

- Input the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)
- The conditions that the value at the specified address of the input register must satisfy.

Waiting for holding register

Description: Wait for the value at the specified address in the holding register to meet a certain condition before continuing execution of the next instruction.

Parameters :

- Hold the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)
- The conditions that must be met to maintain the value at a specified address in a register.

Waiting contact register

Description: Wait for the value at the specified address in the contact register to meet a certain condition before continuing to execute the next instruction. **Parameters :**

- The starting address of the contact register.
- The conditions that the value at the specified address of the contact register must meet.

Waiting coil register

Description: Waits for the value at the specified address in the coil register to meet a condition before continuing execution of the next instruction. **Parameters :**

- Coil register start address.
- The conditions that the value at the specified address of the coil register must meet.

Read input register

Description : Retrieves the value at

the specified address of the input

register.

Parameters :

- Input the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number

(8 bytes, occupying 4 registers) **Return value:** The value at
the address specified in the input register.

Read holding register

Description : Retrieves the value at

the specified address of the

holding register.

Parameters :

- Hold the starting address of the register.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number

(8 bytes, occupying 4 registers) **Return value:** Holds the
value at the specified address of the register.

Read contact register

Description : Retrieves the value at a specified address in the contact register. **Parameter :** The starting address of the contact register.

Return value: The value at the specified address of the contact register.

Read coil register

Description : Retrieves the value at the specified address in the coil register. **Parameter :** The starting address of the coil register.

Return value: The value at the specified address of the coil register.

Continuous reading of coil register

Description : Reads the value at the specified address of the coil register continuously. **Parameters :**

- Coil register start address.
- The number of register bits read sequentially.

Return value: The value at the specified address of the coil register, stored in a table. The first value in the table corresponds to the value at the starting address of the coil registe.

Continuous Read Holding Register

Description : Reads the value of the holding register at the specified address continuously.
parameter :

- Hold the starting address of the register.
- The number of values read consecutively.
- Select the data type you want to read:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)

Return value: The value at the specified address of the coil register, stored in a table. The first value in the table corresponds to the value at the starting address of the coil register .

Write to coil register

Description : Writes the specified value to

the specified address in the coil register.

Parameters :

- Coil register start address.
- The value to be written can only be 0 or 1.

Continuous writing to coil register

Description : Writes the specified value

continuously to the specified address in the coil

register.

Parameters :

- Coil register start address.
- Input the value to be written, with multiple values separated by commas, and each digit can only be 0 or 1.

Write to holding register

Description : Writes the specified value to the holding register at the specified address.
parameter :

- Hold the starting address of the register.
- The value you choose to write must correspond to the data type you select.
- Select the data type to write:
  - U16: 16-bit unsigned integer (2 bytes, occupying 1 register)
  - U32: 32-bit unsigned integer (4 bytes, occupying 2 registers)
  - F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers)
  - F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers)

Shut down the main site

Description : Shut down the Modbus master station and disconnect it from all slave stations.

Bus block group

The bus block group is used to read and write Profinet or Ethernet/IP bus registers. For instructions on using the bus communication function, please refer to the
" CBSH Bus Communication Protocol Document (EtherNet/IP, Profinet)".

> illustrate :
> Magician E6 does not support this set of instructions.

Get the value of the bus register

Description : Retrieves the value

of the specified bus register.

Parameters :

- Select the register type; bus input registers and bus output registers are supported.
- Choose the data type of the register; bool, int, and float are supported.
- Choose the address of the register.

Return value: The value of the specified register. A boolean value can be 0 or 1.

Set the value of the bus register

Description : Sets the value of

the specified bus register.

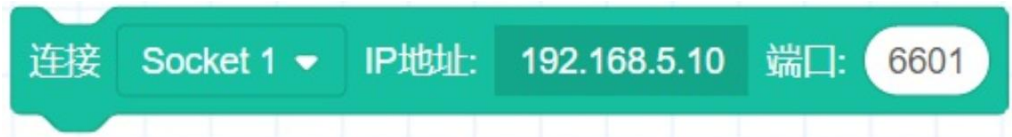
Parameters :

- Select the register type; currently only bus output registers are supported.
- Choose the data type of the register; bool, int, and float are supported.
- Choose the address of the register.
- Enter the value you want to set. A boolean value can be either 0 or 1.

TCP building block set

The TCP block group is used for TCP communication-related operations. Refer to the corresponding demo for a quick experience of the relevant commands.

Connect SOCKET



Description : Creates a TCP client to communicate

with a specified TCP server. **Parameters :**

- Select the SOCKET number to establish a maximum of 4 TCP communication links.
- The IP address of the TCP server.
- The port of the TCP server.

Get connection SOCKET result

Description : Retrieves TCP

communication connection

results. **Parameter :** Select

SOCKET number.

Return value: 0 for a successful connection, 1 for a failed connection.

Create SOCKET

Description : Creates a TCP server and

waits for clients to connect.

Parameters :

- Select the SOCKET number to establish a maximum of 4 TCP communication links.
- The IP address of the TCP server.
- The port for the TCP server. Do not use the ports listed below that are already in use by th e system, as this will cause server creation to fail.

  7, 13, twenty two, 37,

  139, 445, 502, 503,

  1501, 1502, 1503, 4840, 8172, 9527,

  11740, 22000, 22001, 29999, 30004, 30005, 30006,

  60000~65504, 65506, 65511~65515, 65521, 65522

Get SOCKET creation result

Description : Retrieves the TCP

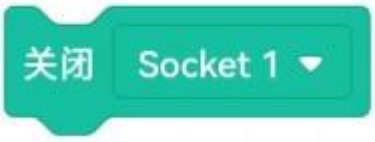
server creation result.

Parameter : Select the

SOCKET number.

Return value: 0 if creation is successful, 1 if creation fails.

Close SOCKET

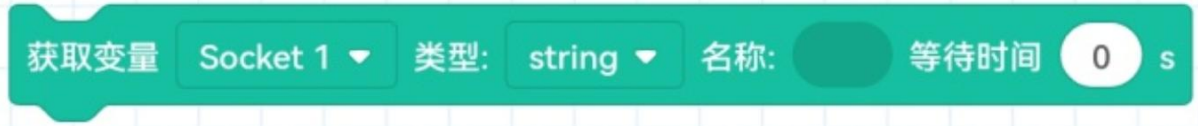


Description : Closes the specified socket,

disconnecting the communication link.

Parameter : Select the socket number.

Reading variables



Description : Reads and saves

variables via TCP communication.

Parameters :

- Select SOCKET number.
- Choose the type of variable to receive; strings and numbers are supported.
- The variable used to store the received data uses the already created variable blocks.
- Enter the timeout period. If set to 0, there will be no timeout, and the system will wait inde finitely until the variable is received.

Get the result of reading the variable

Description : Retrieves the

result of TCP read variables.

Parameter : Select the

SOCKET number.

Return value: Returns 0 if the read is successful, and 1 if the read fails.

Send Variables

Description : Sends variables

via TCP communication.

Parameters :

- Select SOCKET number.
- The data sent can be filled in directly or other oval blocks that return strings or numbers.

Get the result of sending variables

Description : Retrieves the

results of TCP sent variables.

Parameter : Select the

SOCKET number.

Return value: 0 if the transmission is successful, 1 if the transmission fails.

Tray building blocks

A pallet is a container for holding batches of materials according to a set pattern, commo nly used in automated loading and unloading processes. A pallet typically contains an arr ay of recesses, each capable of holding one material. Using pallet commands, a complet e pallet array can be created by teaching a small number of points , and the specific loca tions within the created pallet can be retrieved, enabling rapid automated loading and unl oading by the robot.

Create a tray

Description: Create trays. Supports creating 1D, 2D, and 3D trays. A maximum of 20 tra ys can be created. Creating a tray with the same name will overwrite an existing tray an d will not increase the tray count.

parameter :

on the block , then click the block to bring up the settings

window and configure the pallet parameters. First, select the

pallet dimension; different dimensions of pallets require different

parameters.

- One-dimensional tray

A one -dimensional tray is a set of points equidistantly distributed along a straight line.

  - First, fill in the number of objects P1~P2, which is the total number of points on the one-dimensional pallet.
  - Then configure points P1 and P2 separately, which can only be selected from the stored point list.

- Two-dimensional tray

A two-dimensional tray is a set of points arrayed on a plane.

- Fill in the number of objects in rows P1~P2 and columns P1~ P4 respectively . The product of the two is the total number of points on the two-dimensional tray .
- Then configure points P1 to P4 respectively, which can only be selected from the stored point list.

- 3D tray

A three-dimensional pallet is a set of points distributed in three dimensions in space, which can be regarded as multiple two-dimensional pallets arranged vertically.

- Enter the number of objects in rows P1~P2, columns P1~P4, and layers P1~P5 respectively. The product of these three numbers is the total number of points in the 3D tray .
- Then configure points P1 to P8 respectively, which can only be selected from the stored point list.

Notice :

If you are using an end-effector tool, make sure that the tool coordinate system corresponding to the end-effector tool is selected when teaching the point.

Get the total number of tray points

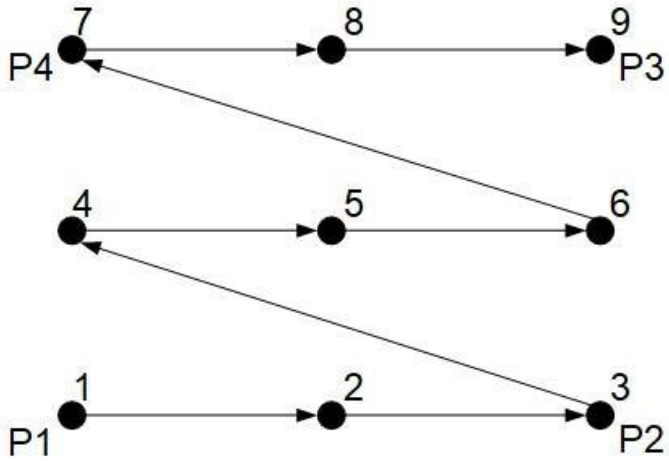
Description : Retrieves the total number of locations for created pallets. **Parameter :** Select the name of the created pallet.

Returns : The total number of points on the specified pallet.

Get tray point

Description : Retrieves the position with a specified index on a specified tray. The index number depends on the order of the positions passed in when the tray was created.

- One -dimensional tray: Point P1 is numbered 1, point P2 is numbered the same as the number of points, and so on.
- Two-dimensional tray: The following diagram uses a 3x3 tray as an example to illustrate the relationship between teaching points and point numbers.



- Three-dimensional tray : Similar to the two-dimensional tray, the index of the first point of the second layer is the index of the last point of the first layer plus one, and so on .

parameter :

- Select the name of the created tray.
- Enter the index of the

point you want to retrieve.
Returns : The coordinates of the point with the corresponding index.

Quick Experience

- Reading and writing Modbus register data
- Data is transmitted via TCP communication.

Reading and writing Modbus register data

Scene Description

To experience how to read and write Modbus data through graphical programming, let's assume we want to implement the following scenario:

The robot creates a Modbus master station, connects to an external slave station, and reads the address of a specified coil register. If the value of that address is 1, the robot moves to point P1.

Programming steps

To achieve the above scenario, the program we need to write is shown in the figure below.

2. Create a master site with the same IP address as the slave site. Keep the default values for port and ID. For quick verification, we've used the robot's built-in slave site, so its IP address is the robot's address (default 192.168.5.1, can be modified).

3. The program checks whether the main site was created successfully. If it was created su ccessfully, the subsequent steps will be executed; otherwise, the program will terminate di rectly.
4. If the value of the robot coil register 9 has been modified, it may affect the subseque nt program logic. Therefore , the value of the coil register 0 needs to be set to 0 first.
5. Wait for the value of coil register 9 to become 1.
6. Control the robot to move to point P1, where P1 is a user-defined save point.
7. Shut down the main site.

If you want to connect to a third-party slave station, please modify the IP address and po rt in the master station block to the address of the third-party slave station. For the range and definition of register address values when reading and writing registers, please refer to the Modbus register address definition description of the corresponding slave station.

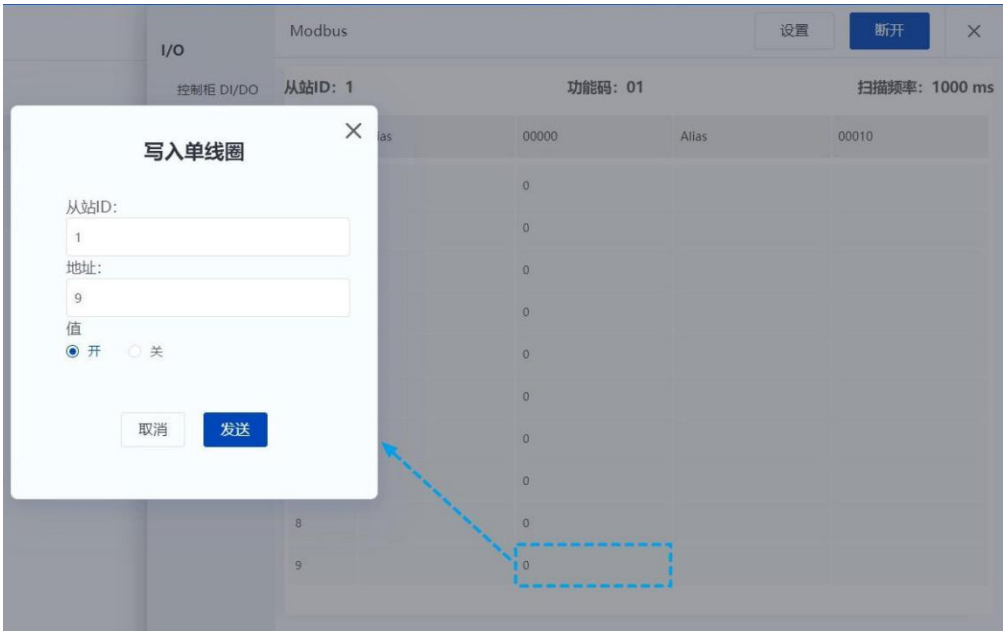
Run the program

If you need to run the program quickly, you can use **the Modbus monitoring** tool to modify the value of the coil register.

2. Open **monitoring > On the Modbus** page, click **the connection in the upper right corner** .
3. The default connection settings are shown in the image below. No modifications are nee ded; simply click **to connect** .

4. After the Modbus master station is successfully created, double-click (PC) or click (mobile) the cell corresponding to the value of coil register 9 to bring up **the Write Single Coil** window.

5. Change the coil **value** to **"on"** and click **"send"** .



6. Observe whether the robot moves to P1.

Data is transmitted via TCP communication.

Scene Description

To experience how to perform TCP communication through graphical programming, let's assume we want to implement the following scenario:

The robot creates a TCP server, waits for clients to connect and send the "go" command, and then returns "Go". to The message "P1" is sent and the movement begins to point P1.

Programming steps

To achieve the above scenario, the program we need to write is shown in the figure below.

2. Create a TCP server (Socket) 1) The IP address is the robot's address, and the port is customizable.
3. The program checks whether the server creation was successful. If the creation was successful, the subsequent steps will be executed; otherwise, the program will terminate directly.
4. Wait for the client to connect and send a string, then save the received string to the string variable tcp. In `rec v`, string variables need to be created by the user.

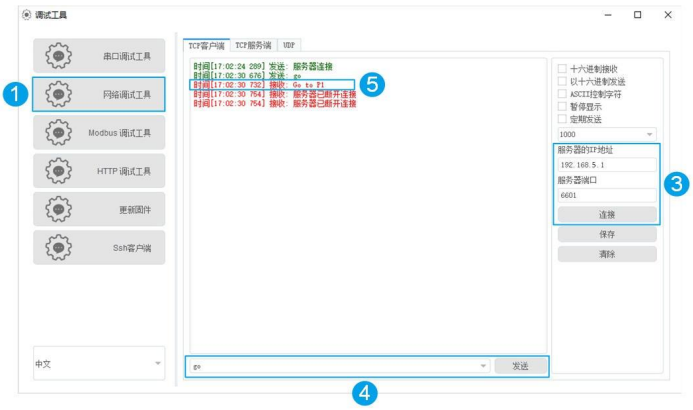
5. Determine if the received string contains "go". If it does, proceed to steps 5 and 6; otherw ise, proceed directly to step 7.
6. Send the string "Go" to P1 is given to the client.
7. Control the robot to move to point P1, where P1 is a user-defined save point.
8. Shut down the TCP server.

Run the program

If you need to run the program quickly, you can use CBSH Studio. The Pro's built-in debuggin g tools act as a TCP client.

2. Open **Settings > Debugging tools** , enter **network debugging tools > TCP client** page.
3. the robot to a point outside of P1 (to facilitate subsequent observation of whether th e robot has executed the movement command), then save and run the program .
4. After the runtime log indicates that the TCP server has been created successfully, modify the server's IP address and port in the debugging tool, and then click Conn ect .
5. the connection is successful, type "go" below the debugging tool and click send.
6. Check if the debugging tool received "Go" to The message

"P1" indicates whether the robot has moved to P1. The image below shows the debugging tool interface, with the numbers corresponding to the steps described above.



Appendix C Script Programming Function Description

Basic Syntax

Basic Concepts

illustrate :
If you wish to systematically learn Lua programming, please search for Lua tutorials online. This manual only lists some basic Lua syntax for your quick reference.

identifier

Identifiers are used to define a variable, function, or other user-defined item. Identifiers b egin with a single letter. A arrive Z or a arrive z or underline _Add zero or more letters, un derscores, and numbers (0 to 1) after the beginning. 9).

It's best to avoid using underscores and uppercase letters as

identifiers, because Lua reserved words also use this convention.

Lua does not allow special characters such as **@** . **$** , and **%** Use

this to define the identifier.

Lua It is a case-sensitive programming language, and the case of all identifiers in the program must be consistent with the examples provided in this manual .

Keywords

The following are listed Lua Reserved keywords. Reserved keywords cannot be used as cons tants, variables, or other user-defined identifiers:

and, break, do, else, elseif, end, false ,for, function, if , in , local, nil, not, or, repeat, return, then, true, until, while, goto

As a general convention, a name beginning with an underscore and followed by a string of uppercase letters (e.g., ...) is used. _____VERSION) is reserved for Lua Inter nal global variables .

Notes

Comments do not affect the execution of the program; they are mainly used to help people w ho read the code understand it.

Single-line comment

In a single line of code, everything following the two hyphens is considered a comment. Com ments can be on a separate line or follow the code.

```
--Single -line comment
print ( "Hello World ! " ) --Single -line comment
```

Multiline comments

Multiline comments begin with "--[[" and end with " The text ends with "--" and the content b etween the two is considered a comment.

```
-- [[
  Multiline
  comment
  s
  Multiline
  comment
  s
  -- ]]
```

Variables and Data Types

Variables are used to store values, which are then passed as parameters or returned as resul ts. Variables are assigned values using the "=" sign.

In Lua, use local Explicitly declared variables are local variables (effective within the curr ent function). The scope of a local variable extends from its declaration location to the e nd of the function it belongs to.

Unused local Variables declared explicitly are script-level variables by default (effective in the current script file), and the scope of script-level variables is a single script file.

In addition, you can also use CBSH Studio Pro's **monitoring > The global variables** page sets global variables at the controller level, which can be directly accessed in different s cript files within the same controller.

Example 1 : This mainly illustrates the difference in scope between local variables and script-level variables.

```
function func () local      -- Define a function that
    a = 1 b = 2             prints a and b when run.
                            -- Local variables
                            -- script-level variables

en d

f unc()  pr                 -- The function is executed, printing the values of a and b as 1 and 2 respectively.
int (a) pr                  -->  nil ( when called outside the variable's scope, it is
int (b)                     printed as nil , the same applies below )
                            --> 2
```

Example 2 : This mainly illustrates that local variables can have the same name as scrip t-level variables, but they only take effect within the function. Code blocks that control fl ow (loops , conditional statements), such as do/end, are also considered functions.

```
a = "a"


for i=10,1,-1 do do
        local a = 6
        print(a)                    -- Local variables
end                                 --> 6. The variable ' a' is called within the statement block .

print(a)
                                --> a , the script-level variable called outside the statement block .
```

Example 3 : This mainly illustrates the difference between the scope of global variables and t he other two types of variables, as well as the global retention function of global variables.

```
-- The src0.lua file of Project 1
-- Assuming two global variables , g1 and g2, have
been added to the global variables page...
-- g1 is not globally persistent and has a value of 10.
-- g2 is globally persistent and has a value of 20.

local a = 1 b =
2
print(a)                --> 1
print(b)                --> 2

print(g1)                 --> 10
print(g2)                 --> 20
SetGlobalVariable("g1",11)              - -  to global variables that are not gl
                                             obally persistent
SetGlobalVariable ( "g2" , 22 ) p ri    - -  to global variables that are main
nt (g1)                 --> 11               tained globally
p ri nt (g2)                --> twenty
two

-- The src0.lua file of
Project 2
-- Project 2 runs after Project
1

pr int (a) pr           --> nil
int (b) pr int          --> nil
(g1) pr int             -->   10 ( Non-globally persisted variables were restored to
(g2)                    their values before the modification in Project 1 )
                        --> 22 ( The globally retained variable has been changed to
                        the value modified in Project 1 )
```

Variable names can be any string consisting of letters, underscores, and numbers that do not begin with a number, except for Lua-reserved keywords.

Lua Variables do not require type definitions; simply assigning a value to a variable will a utomatically determine its type based on the value. Assigning a value of a different type t o an already assigned variable within a script will change the variable's type; however, in **monitoring... > The type of** variables set in **the global variable page cannot be changed** , and the type of the assigned value must be consistent with the type selected when the v ariable was created; otherwise, an error will occur when the controller runs the script.

Lua supports various data types, the most common of which include numbers, booleans, strings , and tables. Arrays in Lua are a type of table.

Lua also has a special data type called nil. This indicates an empty string (without any v alid value). For example, printing an unassigned variable will output an empty string. nil value.

number

In Lua, the number is a double-precision floating-point number that supports various operatio ns. The following notations are all treated as number:

- 2
- 2.2
- 0.2
- 2e+1
- 0.2e-1
- 7.8263692594256e-06

Boolean value

boolean The type has only two possible values: true. and false (fake), Lua Bundle false an d nil Seen as False, all others are true, numbers 0 Too true.

String

A string is a sequence of characters consisting of numbers, letters, and underscores. A string can be represented in the following three ways:

- A string of characters between single quotes.
- A string of characters between double quotes.
- **[[** and **]]** A string of characters.

When performing arithmetic operations on a numeric string, Lua... It will attempt to convert th is numeric string into a single number.

```
-- Use single quotes to define strings.
local str1 = Hello, Lua!'          print (str1)
- - Output : Hello, Lua!

-- Use double quotes to define strings.
local str2 = Hello, Lua!          print (str2)
- - Output : Hello, Lua!

-- use [[ and ]]       local 定义多行字符串
str3 = [[        This is a
multi-line string in Lua.
]]
pr int (str3)
-- Output :
-- This is a
-- string in          multi-line
                      Lua.
--  Lua Automatically convert numeric strings to numbers and perform
arithmetic operations.                  local numStr = "10"
local result = numStr + 20 -- Automatically "10" Convert to number 10
print (result)  - - Output :  30
```

surface

A tab is a set of indexed data.

- The simplest way to construct a table is by using {}, which creates an empty table. Alterna tively, you can directly initialize the table.
- The table is actually an associative array, which can use values of any type as indices, but these values cannot be of any type. nil.
- The table has a variable size and can be expanded as needed.
- The length of a table can be obtained using the # symbol.

```
-- Initialize a table with a contiguous index.
local tbl = {[1] = 2, [2] = 6, [3] = 34, [4] = 5}
print ( "tbl") length " , #tbl)  --Output :    4
```

In this example, `tbl` Index `[1]` [`4]` It is continuous, therefore `#tbl` Correctly returned length `4` .

array

An array is a collection of elements of the same data type arranged in a certain order; it can be a one-dimensional array or a multi-dimensional array.

In Lua, arrays are of type `table`, and index keys can be represented by integers. The size of an array is not fixed.

- One-dimensional array: The simplest array, whose logical structure is a linear list.
- Multidimensional arrays: These are arrays that contain other arrays or one-dimensional arrays whose index keys correspond to an array. In Lua, array indices default to starting from... It can start from 1, but you can also specify from. 0 Or it can start with a negative number. Accessing an array element using a non-existent index will return nil.

Example 1 : A one-dimensional array can be used to loop through its elements using a for lo op.

```
-- Create a one-dimensional array
local array = { "Lua" , "Tutorial "

-- Use a for loop to iterate through the array, starting from index 1.
for i = 1 , #array do
      p rint ( array[i])      --Output : Lua Tutorial
en d
```

Example 2 : A one-dimensional array containing numeric elements

```
-- Create a one-dimensional array whose internal elements are numbers.
local numbers = { 10 , 20 , 30 , 40 , 50 }

-- Use a for loop to iterate through the array, starting from index 1.
for i = 1 , #numbers do                                    p rint
      (nu mbers[i])      - - Output : 10 20 30 40 50
end
```

Example 3 : Arrays with custom indices

```
-- Create an array and use negative and positive numbers as indices.
 local array = {} for i =
                -2 , 2 do
              array[ i] = i  *  2 + 1 -- 为数组赋值
en d

-- Use a for loop to iterate through the -2 到 2 的索引...
for i = -2 , 2 do            print
      (array[i])    --                  输出: -3 -1 1 3 5
en d
```

Example 4 : A 3x3 array of multidimensional arrays

```
-- Initialize array
array = {}
```

```
for i=1,3 do
  array[i] = {}
      for j=1,3 do
         array[i][j] = i*j
      end
end

-- 访问数组
for i=1,3 do
    for j = 1 , 3 do              pr
        int (array[i][j])                --打印结果分别为：1 2 3 2 4 6 3 6 9
    en d
en d
```

Operators

Arithmetic operators

Instruction symbols	illustrate
+	addition operation
-	Subtraction
*	Multiplication operations
/	Floating-point division operation
//	Floor division
%	Remainder division operation
^	Exponentiation

Example :

```
a = 20
b = 5
prin t (a+b)                --打印a加b的结果：25
prin t (ab) prin            --打印a减b的结果：15
t (a*b) prin t              --打印a乘b的结果：100
(a/b) pr int                --打印a除以b的结果：4
(a//b) prin t                --打印a整除b的结果：4
(a%b) pr int                --打印a除以b的余数结果：0
(a^b)                       --打印a的b次幂的结果：3200000
```

Bitwise operators

Instruction symbols	illustrate
&	Bitwise AND operation
\	Bitwise OR operation
~	Bitwise XOR operation
<<	Bitwise left shift operation
> >	Bitwise right shift operation

Example :

```
pr int (a&b)                --打印a和b按位与的结果：4
```

```
prin t (a|b)                Print the result of the bitwise
prin t (a~b) pr             OR operation between a and b :
int (a<<b) pr               21
int (a>>b)                  --Print the result of XORing a
                            with b : 17
                              --Print the result of shifting 'a' to the left by ' b ' units: 640
                              --Print the result of shifting 'a' to the right by ' b ' units: 0
```

Relational operators

Instruction symbols	illustrate
= =	equal
~ =	Not equal to
< =	Less than or equal to
> =	Greater than or equal to
<	Less than
>	Greater than

Example :

```
a = 20                      --Create variable
b = 5                         a
pr int (a==b) pr              --Create
int (a~=b) pr                 variable b
int (a<=b) pr                 --Print the comparison result of a equal
int (a>=b) prin               to b : false
t (a<b) pr int                --Print the comparison result of a not
(a>b)                         equal to b : true
                              --Print the comparison results of a being
                              less than or equal to b : f alse
                              --Print the comparison result of a being
                              greater than or equal to b : true
                            --Print the comparison result of a
                            being less than b : false
                            --Print the comparison result of a
                            being greater than b : true
```

Logical operators

Instruction symbols	illustrate
a nd	The logical AND operator . The result is true only if both sides are true; if either side is false, the result is false.
or	The logical OR operator, when either side returns true, The result is true; only if both sides are false, the result is false.
n ot	The logical NOT operator directly negates the result of the comparison.

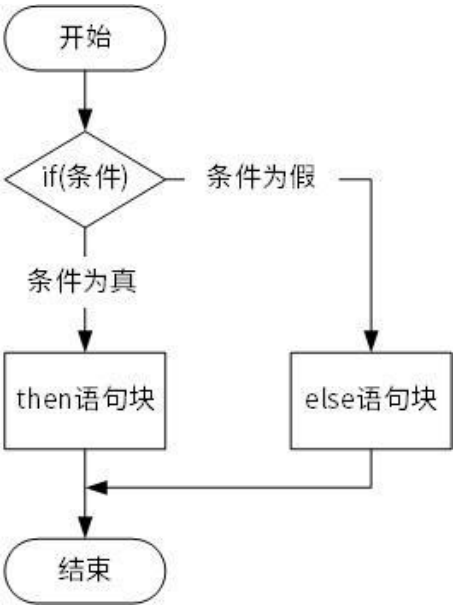
```
a=true
b=false
print(a and b)          --Output :  The function returns true only if both a and b are true .
print(a or b)           -- Output :  true , as long as either a or b is true. It returns true.
print(not a)            --Output :  `false` means `a` is true , and its negation is false.

print(not  b)           -- Output :  If b is true , b is false , and the negation is true.
print(not (20>5))       --Output :  False , 20>5 is true , and the negation is false.
```

Instruction symbols	illustrate
if… then… els e…end	`if` conditional statement evaluates each condition sequentially from t op to bottom. If a condition is true , the corresponding code block is e xecuted, and subsequent condition checks are ignored.
wh ile… d o …end	The while loop control instruction. When the condition is... true This allo ws the program to repeatedly execute certain statements. Before exec uting a statement, it first checks if the condition is true. true
for r…do… e nd	The `for` loop directive repeatedly executes a specified statement, with th e number of repetitions allowed. for Control in statement
re peat… u n til()	The `repeat` loop control instruction. It repeats the loop execution until... U ntil the specified condition is true.

if conditional statement

The parentheses following the `if` statement contain a conditional expression, the result of which can be any value in Lua. Bundle false and nil Consider it false, and everything else true; numbers. 0 It is also true. When the expression is true, the then statement block is executed; when the expression is false, if there is an else statement block, the else statement block is executed ; otherwise, the statements after end are executed directly.

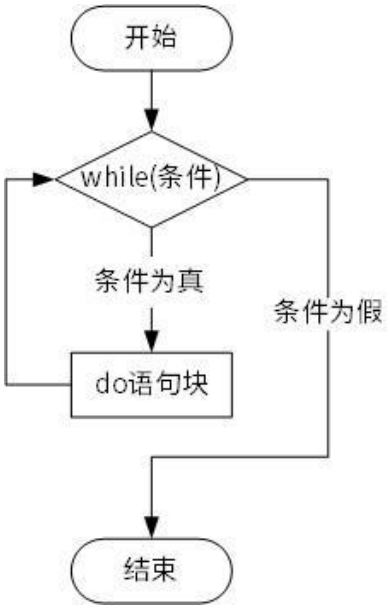


If statements can be nested; below is a typical example.

```
a = 100 ; b
= 200 ;
--[ Check conditions] --]
if (a == 100 ) th
en
    -- [The following if condition is executed when the if condition is true-- ]
    if (b == 200 ) th
    en
        -- [ This block of statements is executed when the if condition is true-- ]
        prin t ( "a The value is : " , a ) -- a The value is :
        100 prin t ( "b The value is : " , b ) -- The value of
        b is : 200
    en d
el se
    -- [ The following block of statements is executed
    when the first if condition is false-- ] pr int ( "a is not
    equal to 100" )
en d
```

while cyclic control command

following the while loop contain the conditional expression. If the expression is true, the do statement block is executed, and then the condition is re-evaluated; if the expression is false, the statement after end is executed directly.



Example :

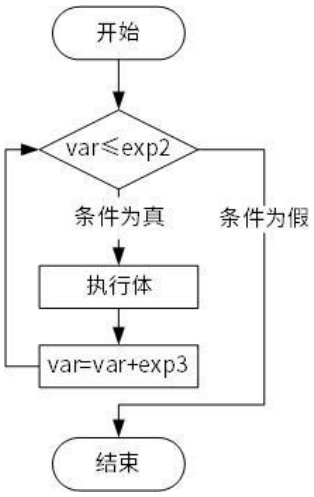
```
a = 10
while ( a < 20 do
      print ( "a The value is : " , a)  -- Execute 10 times, output values
      from 10 to 19. a = a+ 1
en d
```

for loop control instructions

The syntax of the for loop is as follows:

```
for  var=exp1,exp2,exp3  do
      <executor>
en d
```

the variable var is exp1. After each execution of the <executive body>, var is incremente d by exp3 (the value of exp3 can be negative or not specified , and the default value is 1) until var is greater than exp2.

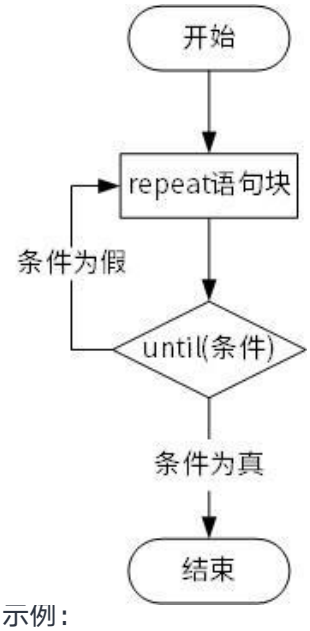


Example :

```
for i = 10 , 1 , -1 do
      print (i) --          执行10次，输出值为10到1
en d
```

re peat loop control command

The repeat loop is similar to the while loop, but the main difference is that while the cond ition is checked before the statement to be looped is executed, and the loop continues if the condition is true ; while repeat the condition is checked after the statement to be lo oped is executed, and the loop continues if the condition is false.



示例：

```
a = 10
repeat
      print("a的值为:", a)  -- a = a +  Execute 5 times, output value is 10 to 15
      1
until(a > 15)
```

function

Functions are the primary method for abstracting statements and expressions, and their defin ition format is as follows:

```
function  function_____name (argument1,  argument2,  argument3 ... , argumentn)
      function       _bo dy
      return  result_____p arams_c omma_se parated
en d
```

- **fun ction_name** : Function name. A function can be defined first and then called by its name, or it can be defined directly when called. In the latter case, the function name can be omitted.
- **argument1, argument2, argument3 ... ... argumentn:** Function parameters, multipl e parameters are separated by commas . Functions can also be without paramet ers.
- **function _body:** The function body is the block of code statements that need to be

executed within the function.

- **res ult_____pa rams_co mma_separated:** Function return values: Lua functions can return multiple values, each separated by a comma. Functions can also have no return value.

Example 1 : Functions without input parameters and return values

```
function greet ()
    print ( "Hello,  Lua !  - - function body
en d

greet() -- Calls a function that outputs : Hello, Lua!
```

Example 2 : Functions with input parameters but no return value

```
function printSquare (number)
    print ( "Square") of " ...    number .. " is " . (number  *  number))  -- function body
end

printSquare(5)   --          调用函数，输出: Square of 5 is 25
```

Example 3 : Functions with input parameters and return values

```
function maximum (a)
    local  mi  =  1  - - Maximum index
    local m = a[mi] -- Maximum value
    for i, val in ipairs (a) do
        if val > m then
            mi = i m
            = val
        en d
    en d
    return  m,  mi  - - Return maximum value and index
en d

local maxVal, maxIndex = maximum({8, 10, 23, 12, 5}) print("Max
value:", maxVal) -- 输出: Max value: 23
print ( "Index") of max value:" , maxIndex) -- Output : Index of max value: 3
```

illustrate :

This manual primarily introduces how to use Yuejiang's predefined functions. Users can also define their own functions. If you need to define a custom function, please write the function definition in global.lua or at the top of the src*.lua file that calls the function; otherwise, a runtime error will occur.

General functions for mathematical calculation

Lua provides some basic mathematical functions as a supplement to arithmetic operators.

m ath.abs(X)

Returns the absolute value of X, for example:

```
p rint ( "math. abs(-10):" , m ath .a bs( -10 ))   - - Output : 1 0
```

m a th.floor(X)

Returns the largest integer value not greater than X (rounded down), for example:

```
print ( "math.floor(3.7):" ,    math.floor ( 3.7 ))   --                  输出: 3
```

m ath.ceil(X)

Returns the smallest integer value not less than X (rounded up), for example:

```
p rint ( "math.c eil(3.2):" , m ath .ce il( 3.2 ))   --                  输出: 4
```

m ath.sqrt(X)

Return the square root of X, for example:

```
p rint ( "math. sqrt(16):" , m ath .s qrt( 16 ))   - - Output : 4
```

m ath.rad(X)

Returns the radian value of angle X (angle to radians), for example:

```
p rint ( "math. rad(180):" , m ath .r ad( 180 ))   - - Output : 3.14159 26535898
```

m ath.deg(X)

Returns the angle value in radians X (radians converted to degrees), for example:

```
p rint ( "math.deg (math.pi):" , m ath .deg( math .pi))   - - Output : 1 80
```

m ath.sin(X)

Returns the sine value of X radians.

If you want to use the angle as a parameter, you can use math.rad to convert it, for example:

```
p rint ( "math.sin(ma th.rad(30)):" , m ath .sin( mat h .rad( 30 )))   --                输出: 0.5
```

m ath.cos(X)

Returns the cosine value of X radians.

If you want to use the angle as a parameter, you can use math.rad to convert it, for example:

```
p rint ( "math.cos(ma th.rad(60)):" , m ath .cos( mat h .rad( 60 )))   --                输出: 0.5
```

m ath.tan(X)

Returns the tangent of X radians.

If you want to use the angle as a parameter, you can use math.rad to convert it, for example:

```
p rint ( "math.tan(ma th.rad(45)):" , m ath .tan( mat h .rad( 45 )))   --                输出: 1
```

m ath.asin(X)

the arcsine of X in radians, which can be converted to degrees using math.deg. For example:

```
print ( "math.deg(math.asin(0.5)):" ,    math .deg( math .asin( 0.5 )))    - - Output : 30 , return value is the angle.
```

m ath.acos(X)

the arccosine of X in radians, which can be converted to degrees using math.deg. For exampl e:

```
print ( "math.deg(math.acos(0.5)):" ,    math .deg( math .acos( 0.5 )))    - - Output : 60 , return value is the angle.
```

m ath.atan(X)

the arctangent of X in radians, which can be converted to degrees using math.deg. For exam ple:

```
p rint ( "math.deg(ma th.atan(1)):" , m ath .deg( mat h .atan( 1 )))   --                输出: 45，返回值为角度
```

matth.log (X)

Return the natural logarithm of X, for example:

```
print ( "math.log(10):" ,  math.log ( 10 ))  --
```

m ath.exp(X)

Returns the natural constant e raised to the power of X, for example:

```
p rint ( "math . exp(1):" , m ath . exp( 1 ))  --
```

General string processing functions

Lua lt provides general functions for string processing, such as searching and replacing strin gs.

string.sub(s, i, j)

Used to extract substrings.

Required parameters

- s: The string to be extracted.
- i: The starting position for the cutoff, starting from 1.

Optional parameters

- j: The position to end the subtraction, the default value is -1, which means the last charac ter.

return

Extract the obtained string.

Example

```
sub1 = string .sub( "abcde" , 3 ) --Starting from the 3rd position: cde
sub2 = string.sub ( "abcde" , 1 , 3 ) -- Extracting from the 1st character to the 3rd
character: abc sub3 = string.sub ( "abcde" , 3 , 3 ) --Extracting the 3rd character: c
```

string.find (s, sub, i, plain)

specified string and return the index of the found substring.

Required parameters

- s: The string to be searched.
- sub: The substring to search for.

Optional parameters

- i: The starting position for the search, which defaults to 1.
- plain: Whether to use plain text mode. The parameter i must be specified when specifying this parameter.
  - true: Use plain text. In this case, sub will be treated as a plain text string.
  - false: The default value. In this case, sub supports pattern matching .

return

- The start and end indices of the first matched substring within the original string.
- If a capture is defined in the pattern, the captured value will be returned after two indices.

- Returns nil if no substring is found.

Example

```
i,j = string .find( "abcde" , "cd" ) --Search for "cd" in the sequence abcde , return indices: i = 3 , j = 4

if string.find (str1, str2) ~= nil --If str1 contains str2 , then execute the contents of TODO .
    -- TODO
    en d

--Advanced usage
i,j = string .find( "abcabc" , "ab" , 3 ) --Search for "cd" in the sequence abcde , starting from the 3rd position, and return the index: i = 4 , j = 5.

i,j  =  string.find ( "123%abc" ,  "%a" )  -- By default, pattern matching is supported. The parameter %a is interpreted as a wildcar d (representing any character), so it will match the first letter ' a' in the original string. i and j are both 5.
i,j  =  string.find ( "123%abc" ,  "%a" ,  1 ,  true )  --Using plain text mode, the parameter %a is treated as plain text, so it will match  %a in the original string , i is 4 , j is 5 .

i,j,sub  =  string.find ( "abc") ;  10  edf  100" ,  "(%d+)" )  --Find and capture the first string of numbers, returning the index of the f ound number and the captured result : i is 5 , j is 6 , sub is 10
```

string.match(s, sub, i)

Search for a substring within a specified string and return either the captured result of the pa ttern match or the substring .

Required parameters

- s: The string to be searched.
- sub: The substring to search for, supporting pattern matching mechanisms .

Optional parameters

- i: The starting position for the search, which defaults to 1.

return

- Returns the matched substring.
- If captures are defined in the pattern, return all capture results.
- Returns nil if no substring is found.

Example

```
sub1 = string .match( "abcde" , "cd" ) --Search for "cd" in "abcde" and return the found substring: cd

sub2 = string.match("abc 10 结果为 :  edf  edf 100" , "%a+" , 4 ) --%a+ indicates a consecutive letter sequence, starting the search from the 4th le tter, and returns the matching letter.

sub3  =  string.match ( "abc  10  edf  100" ,  "%a+  (%d+)  %a+" )  --Captures the number between two strings of letters, returning the captured result: 10
```

string.gmatch(s, sub)

through a specified string to find a substring and return either the captured result or the subst ring that matches the pattern.

- s: The string to be searched.
- sub: The substring to search for, supporting pattern matching mechanisms .

return

- Returns an iterator function that returns a matching result each time the function is called.
- If no matching value is found, the iteration function returns nil.

Example

```
for word in string .gmatch( "Hello world from CBSH " , "%a+" ) do
      pr int (word)
en d
-- [[
  The output will be to print each word line by line:
  He llo
  wo rld
  f rom
  CBSH
  -- ]]
```

string.gsub(s, find, repl, n)

Used to replace a specified part of a string.

Required parameters

- s: The string to be operated on.
- find: The character to be replaced; sub supports pattern matching .
- repl: The character to be replaced.

Optional parameters

- n: Maximum number of replacements. If not specified, all replacements will be performed.

return

The replaced string and the number of replacements.

Example

```
str, n = string .gsub( "aaaa" , "a" , "z" ); --Replace all 'a's with 'z' , return: str is zzzz , n is 4

str, n = string .gsub( "aaaa" , "a" , "z" , 3 ); --Replace the first 3 ' a's with 'z' , and return: str is zzza , n is 3.

str, n = string .gsub( "a1b2c3" , "%d" , "z" ); --Replace all numbers with 'z' , return: str is azbzcz , n is 3
```

必选参数

Pattern matching mechanism

Lua's pattern matching mechanism is similar to regular expressions and can be used with **string.find. string.gmatch, `string.gsub`, `s trin` , and `g.match`** implement fuzzy matching.

Character classes

Character classes are the basic units of pattern matching, used to represent a set of specific characters. The main character classes are as follows:

- Single character (except `^$()%.[]\*+-?)` outside): Matches the character itself.
- `.`: A single dot indicates a match for any character.
- `%a`: Matches any letter.
- `%c`: Matches any control character (e.g., `\n`).
- `%d`: Matches any number.
- `%l`: Matches any lowercase letter.
- `%p`: Matches any punctuation mark.
- `%s`: Matches whitespace characters (spaces).
- `%u`: Matches any uppercase letter.
- `%w`: Matches any letter/number.
- `%x`: Matches any hexadecimal number.
- `%x` (where x is a non-alphanumeric character): Matching the character 'x' is primarily used to han`%` matching special characters (`^$()%.[]*+-?`). For example, `%%` and match .
- `[ Several Characters ]`: Matches any character class enclosed in square brackets `[]`. For example, `[%w_]` with any letter/number (`%w`) or underscore (`_`) match .
- `[^ Several Characters Class ]`: Matches any character class that is not contained in `[ ]`. For example, `[^%s%p] matches` any non-whitespace and non-punctuation character .

All categories represented by single letters (`%a`, `%c`, etc.). If the letters are changed to uppercase, they all represent the corresponding complement. For example , `%S` It represents all non-whitespace characters.

Pattern Entries

Pattern entries are combinations of character classes and special symbols, used to specify the pattern for matching character classes. Commonly used pattern entries are as follows:

- A single character class matches any single character within that class;
- A single character class followed by an asterisk (`*`) will match zero or more characters of that class. This entry always matches the longest possible string;
- A single character class and a `+`, It will match one or more characters of that class. This entry always matches the longest possible string ;

- A single character class and a `-`, It will match zero c `*` nore characters of that class. and different, This entry always matches the shortest possible string;
- A single character class followed by a question mark `(?)`. It will match zero or one of the c haracters of that class.
- `%n`, where n represents an integer from 1 to 9; This entry matches a substring equal to c apture number n (see **captures below).**

model

A pattern is a sequence of pattern entries. An example is shown below:

- To match dates in **dd/mm/yyyy** format, use the pattern `%d%d/%d%d/%d%d%d%d`.
- To match words that start with a capital letter, use the pattern `%u%l+`.

capture

A pattern can enclose a sub-pattern in parentheses. These sub-patterns are called captures.

When a match is successful, the substring matched by the capture will be saved and re turned later or used for operations. Captures are numbered according to the order of th eir left parentheses. For example, for the pattern `(a*(.)%w(%s*))`:

- The string matches `a*(.)%w(%s*)` The string is "Capture #1" .
- Matche `.` The character " is 2 Capture No. 1
- The string that matches " `%s*` " is 3. Capture No. 1
- Captures #2 and #3 are substrings of capture #1.

As a special case, the empty `()` It will capture the position of the corresponding char acter in the string. For example, if we use the pattern `(aa())` Apply to the string `falaap` Abo ve, two captures will be generated: 3 and 5. .

Other commonly used methods

method	illustrate
string.upper (argument)	Convert all letters in the string to uppercase
string.lower (argument)	Convert all characters in the string to lowercase
str ing.reverse(arg)	String reversal
strin g.format(...)	Returns a formatted string similar to printf.
str_ng.len (arg)	Calculate string length
string.rep(string, n)	Returns n copies of the string.

Example :

```
str = "Lua"
p rint ( string.upper ( str))            --Convert all letters in the string to uppercase, and print the result: LUA
p rint ( string.lower ( str))            -- Convert the string to lowercase, print result: lua
print ( string .r everse(str) )      -- Reverse the string, print result: auL
print ( string . len( " abc" ))           --Calculate the length of the string "abc" , print the result: 3
print ( string.format ( "the")) value  is:  %d" , 4 ))        --Print result: the value is:4
print ( string.rep ( str, 2 ) )           -- The string was copied twice , and the printed result was: LuaLua
```

Other operators

Instruction symbols	illustrate
...	Concatenate two strings
#	Returns the length of the string or table.

```
a = Hello b     =
"World" c = { 1 ,
2 , 3 }
print (a..b) -- Prints the string concatenated from a and b : Hello World

print (#b) --Print the length of string b : 5

print (#c) --Length of print table c : 3
```

General functions for table (array) operations

Lua lt provides general functions for table (array) processing, enabling operations such as table insertion and sorting. The table operated on in the following instructions must be a contiguous unary array, with indices ranging from 1 to n (where n is the array length) by Lua's default.

table.concat (table, sep, start, end)

Concatenates the elements in the table into a string in index order, supporting the specification of delimiters and start/end positions.

Required parameters

- table: The table to be operated on.

Optional parameters

- sep: Separator. The default value is no separator.
- start: Starting position. The default value is 1.
- end: End position. The default value is the length of the array.

return

The string obtained by concatenation . If the start position is greater than the end position, return an empty string.

Example

```
fruits = { "banana" , "orange" , "apple" }

print ( table .concat(fruits)) --Using the default connection method, the printed result is: bananaorangeapple

print ( table .concat(fruits, "," )) --Specify the delimiter to connect the strings; the printed result is: banana,orange,apple

print ( table .concat(fruits, "," , 2 , 3 )) --Specify the delimiter and index for concatenation; print result: orange, apple
```

table.insert (table, pos, value)

Insert the specified element at the specified position in the table .

Required parameters

- table: The table to be operated on.
- value: The element to be inserted.

Optional parameters

- pos: The insertion position. The default value is the length of the array plus one, that is, inserting at the end of the table.

Example

```
f ruits = { "banana" , " orange" , "apple" }
table .insert(fruits, "mango" ) -- Insert print (fruits[ 4 ])
at the end --Print result: mango

table .insert(fruits, 2 , "grapes" )    -- Insert at index 2          print (fruits[ 2 ],
"," ,fruits[ 3 ]) --Print result: grapes, orange
```

table.remove(table, pos)

Remove the element at the specified position in the table and return the removed value.

Required parameters

- table: The table to be operated on.

Optional parameters

- pos: The insertion position. The default value is the length of the array, i.e., inserting at the end of the table.

return

was removed.

Example

```
fruits = { "banana" , "orange" , "apple" }

print ( table .remove(fruits)) --Remove the last element and print the removed element: apple

print ( table .remove(fruits), 2 ) --Remove the second element, print the removed element: orange
```

table.sort(table, comp)

You can specify a sorting method to sort the elements in the table .

Required parameters

- table: The table to be operated on.

Optional parameters

- `comp`: Sorting method. This parameter must be a function that meets the following requirements:

- It can accept two elements from the table as parameters.
- Returns true if the first argument must precede the second argument; otherwise, returns false. The default sorting method uses the Lua standard operator "<".

Example

```
fruits = { "banana" , "orange" , "apple" , "grapes" }

pr int ( " before sorting " )

for k,v in ipairs (fruits) do
    p r int (v)              --Print result: banana orange apple grapes
en d

-- Sort in ascending order
ta ble .sort(fruits) pr int (
" sorted " )
for k,v in ipairs (fruits) do
    p r int (v)              --Print result: apple banana grapes orange
en d

-- Sort in descending order
ta ble .sort(fruits, function (a,b)
    return a > bend
)
pr int ( " Sorted " )
for k,v in ipairs (fruits) do
    p r int (v)              --Print result: orange grapes banana apple
en d
```

General Instructions

Exercise

The motion modes supported by robotic arms can be categorized as follows.

Joint movement

The robotic arm plans the motion of each joint based on the difference between the curre nt joint angles and the joint angles at the target point, enabling all joints to move simultan eously. Joint motion is not constrained by TCP (Tool). Center The trajectory of a Point is g enerally not a straight line.

Joint movement is not restricted by singular positions (see the corresponding hardware m anual for singular position details). Therefore, if there are no requirements for the moveme nt trajectory , or if the target point is near a singular position, it is recommended to use joi nt movement.

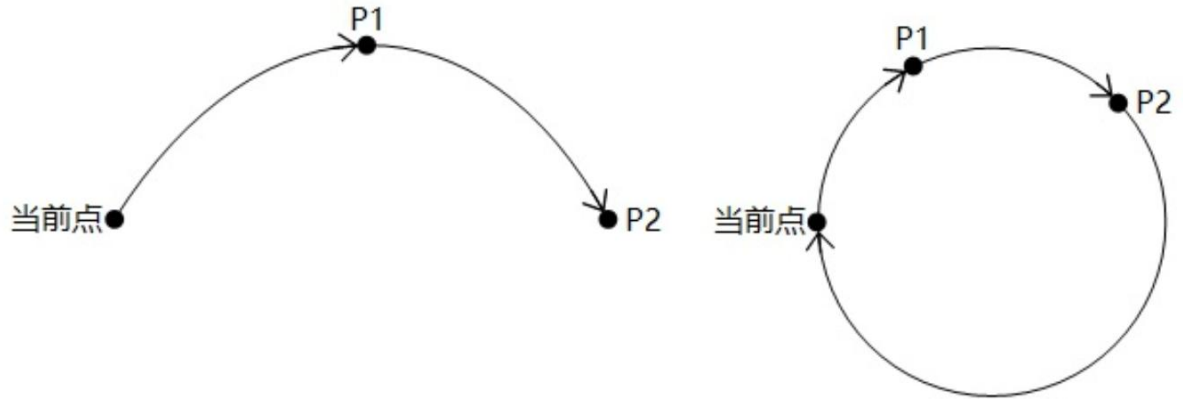
linear motion

The robotic arm plans its motion trajectory based on the current pose and the pose of the target point, making the TCP motion trajectory a straight line, and the end effector's pos e changes at a constant speed during the motion.

When the movement trajectory passes through an odd position, issuing a linear motion co mmand to the robotic arm will generate an error. It is recommended to replan the position or use joint motion near the odd position .

Arc motion

The robotic arm determines an arc or a complete circle using its current position and thr ee non-collinear points, P1 and P2 . The end effector posture of the robotic arm during t he movement is calculated by interpolating the postures of the current point and P2. The posture of point P1 is not included in the calculation (i.e., the posture of the robotic arm when it reaches point P1 during the movement may be different from the taught posture.



When the motion trajectory passes through an odd position, issuing an arc motion comm and to the robotic arm will generate an error. It is recommended to replan the position or use joint motion near the odd position .

Point parameters

Unless otherwise specified, all point parameters in this manual support three expression meth ods:

- Joint variables: The target point is represented by the angles (j1~j6) of each joint of each robotic arm.

  joint variables are used as linear or arc motion parameters, the system will convert the m into pose variables through forward calculus, but the algorithm will ensure that the j oint angles when the robotic arm reaches the target point are consistent with the set values.

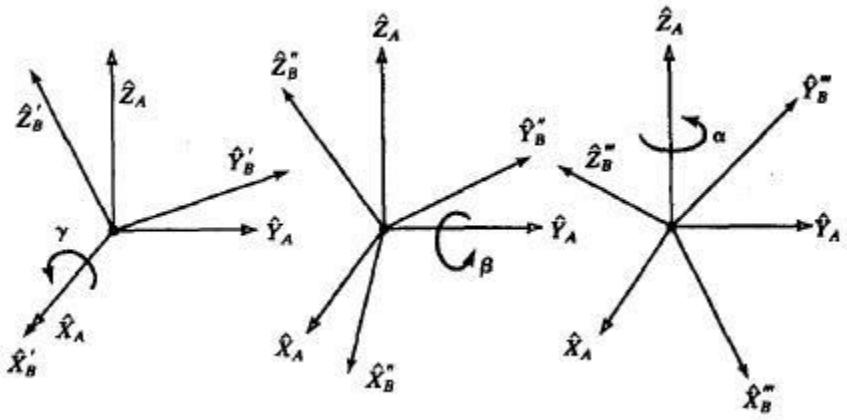
```
{ joint = { j1, j 2, j 3, j 4, j 5, j 6} }
```

- Pose variables: Cartesian coordinates (x, y, z) are used to represent the spatial positi on of the target point in the user coordinate system, and Euler angles (rx, ry, rz) are u sed to represent TCP (Tool). Center The rotation angle of the tool coordinate system re lative to the user coordinate system when the point is reached .

  pose variables are used as point parameters for joint motion, the system will convert t hem into joint variables through inverse kinematics (taking the solution closest to the c urrent joint angle of the robotic arm) for use.

```
{pose = { x, y, z, r x, r y, r z} }
```

the Yuejiang robotic arm calculates Euler angles, the rotation sequence is X->Y->Z, and each axis rotates around a fixed axis (user coordinate system), as shown in the figure be low (rx=γ, ry=β, rz=α).



is determined , the rotation matrix can be used (where cα is short for cosα, sα is short for sin α, and so on).

$$
\begin{aligned}
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
\end{aligned}
$$

Derived into equations

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

通过该方程计算机械臂末端的姿态。

- 示教点位：使用控制软件示教获取的点位会保存为如下格式的常量。

```
--[[                              na
    me：示教点位的名称。
    joint：示教点位的关节坐标。
    tool：示教时使用的工具坐标系索引。
    user：示教时使用的用户坐标系索引。
    pose：示教点位的位姿变量值。
--]]
{
    name = "name" ,
    joint = {j1, j2, tool =    j3, j4, j5, j6},
    index,    user = index,
    pose = {x, y, z,
                                rx, ry, rz}
}
```

Coordinate system parameters

motion commands , user and tool, are used to specify the user and tool coordinate syste ms for the target point. Currently, only index numbers are supported, and the correspondi ng coordinate systems need to be added in the control software first.

The system selects point coordinate systems in the following order of priority:

1. the motion command's optional parameters specify a coordinate system, the specified coordinate system will be used. If the point parameter is a teach point, the pose coor dinates of the teach point will be converted to values in the specified coordinate syste m before use.
2. is not specified via optional parameters:
    - If the point is a teaching point, use the coordinate system index that comes with the teaching point.
    - If the point is a joint variable or pose variable, use the global coordinate system set in the motion parameters (see User and Tool commands for details; the default coordinate system is 0 when the command is not called).

illustrate :

- starts running , the default global coordinate system is set to 0, regardless of the
- value set in the jog panel before running the script. If the target point is a joint variable, the coordinate system parameter is invalid.

Speed parameters

relative speed

The optional parameters 'a' and 'v' are used to specify the acceleration and velocity ratio whe n the robotic arm executes the motion command.

```
actual movement speed of the robotic arm = Maximum speed x
Global Rate x Command rate robotic arm actual motion acceler
ation = maximum acceleration x Instruction rate
```

The maximum velocity/acceleration is controlled by **the reproduction parameters** , which can be found in CBSH Studio . View and modify **motion parameters on the** Pro page.

Global speed can be achieved through CBSH Studio Pro's speed adjustment slider (top right corner of the image above) or SpeedFactor command settings.

The command rate is the ratio carried by the optional parameters of the motion command. When the motion acceleration/velocity ratio is not specified through optional parameters, the value set in the motion parameters is used by default (see VelJ, AccJ, VelL, and AccL commands for details; the default value is used when the command is not called). 1 00).

example :

```
AccJ( 50 ) -- Set the default acceleration for
VelJ ( 60 ) -- joint movement to 50% and the
AccL( 70 ) -- default speed for joint
VelL( 80 ) -- movement to 60%. Set the
              default acceleration for linear
              motion to 70% and the default
              speed for linear motion to 80%.

-- The global rate is 20% .

MovJ(P1) -- (Maximum joint acceleration) x 50% of the acceleration and (maximum joint velocity) x 20% x The joint moves at 60% of its maximum speed to P1. MovJ(P2,{a = 30 , v = 80 }) -- (Maximum joint acceleration) x 30% of the acceleration and (maximum joint velocity) x 20% x 80% of the joints moved to P1

MovL(P1) -- With (the maximum value of C x 70%）的加速度和直线速度（笛卡尔速度最大值 x 20% x 80%）的速度运artesian acceleration moving to P1)
MovL(P1,{a = 40 , v = 90 }) -- 以（笛卡尔加速度最大值 x 40%）的加速度和（笛卡尔速度最大值 x 20% x 90 % ) moves linearly to P1 at a speed of %)
```

Absolute speed

the linear and arc motion commands specifies the absolute speed at which the robotic arm e xecutes the motion command.
Absolute speed is not affected by global speed, but is limited by the maximum speed in **th e reproduction parameters** (if the robotic arm enters the reduction mode, it is limited by t he maximum speed after reduction). That is, if the target speed set by the speed parame ter is greater than the maximum speed in the reproduction parameters, the maximum sp eed shall prevail.

example :

```
MovL(P1,{speed =1,000        - - Move linearly to P1 at an absolute speed of 1000 mm/s
```

If MovL sets the speed to 1000, which is less than the maximum speed of 2000 in the reproduction parameters, the robotic arm will move at a target speed of 1000 mm/s, regardless of the current global speed. However, if the robotic arm is in reduction mode (assuming a reduction rate of 10%) , the maximum speed becomes 200, which is less than 1000, and the robotic arm will move at a target speed of 200 mm/s.

`speed` and `v` parameters are mutually exclusive; if both exist, `speed` takes precedence.

Smooth transition parameters

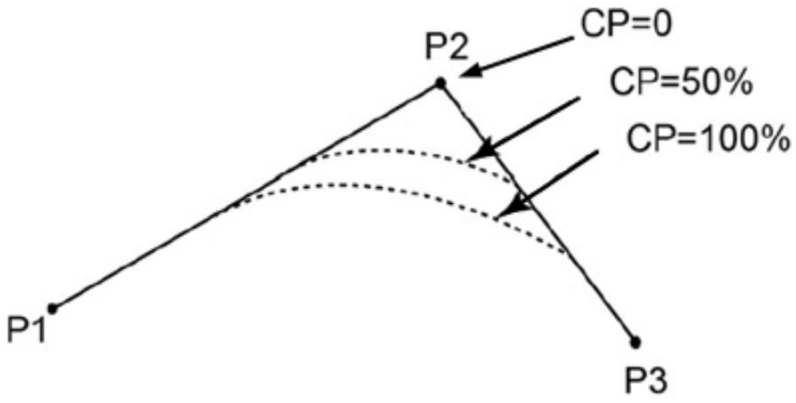
a robotic arm moves continuously through multiple points, it can smoothly transition through intermediate points, avoiding abrupt turns. However, if the user -specified path points are based on different tool coordinate systems, a smooth transition is not possible.

The optional parameters cp or r are used to specify the smooth transition ratio (cp) or sm ooth transition radius (r) between the current motion command and the next motion com mand. **When r is set, the parameter cp is ignored** .

> **illustrate :**
> Joint motion -related commands do not support setting the smooth transition radius (r). Ple ase refer to the optional parameters for each command for details.

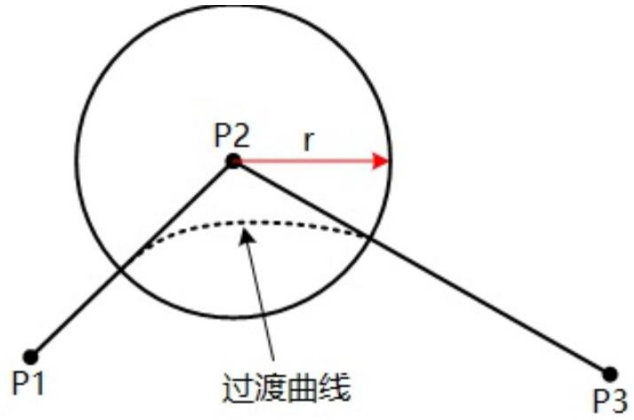
C P

setting the smooth transition ratio, the system automatically calculates the curvature of the transition curve. The larger the CP value, the smoother the curve, as shown in the figure below. The CP transition curve is affected by the motion speed/acceleration. Even if the location and CP value are the same, the curvature of the transition curve will be different when the motion speed/acceleration is different.

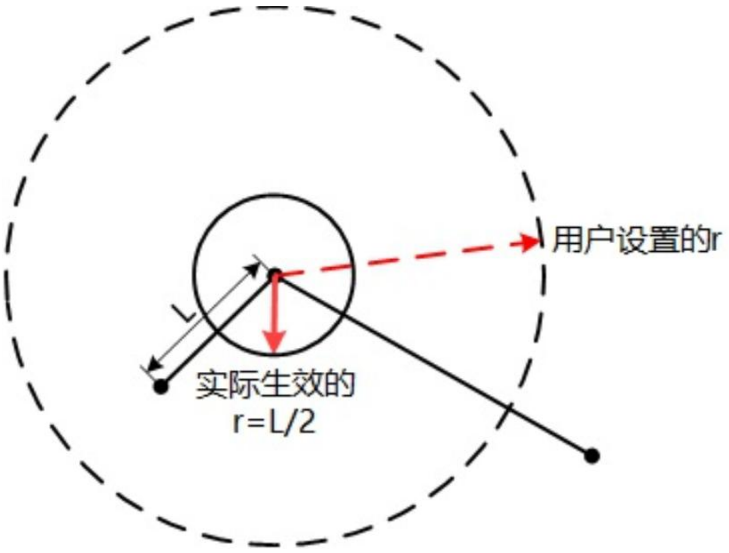


R

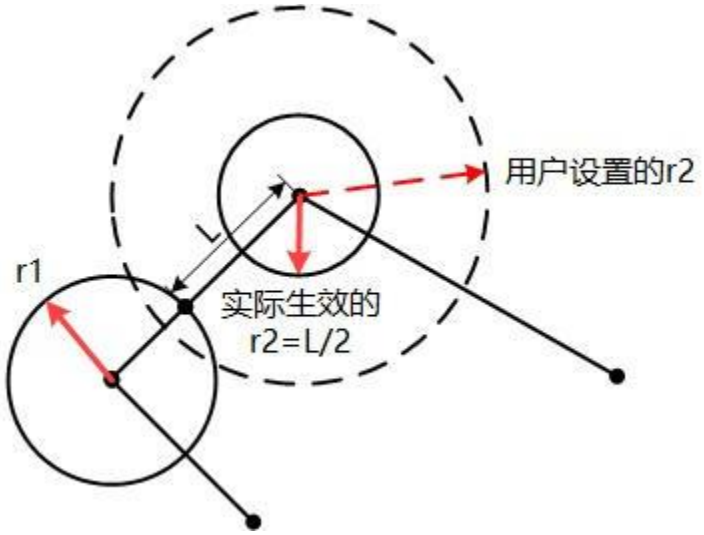
setting a smooth transition radius, the system calculates a transition curve with the transiti on point as the center and the specified radius. The R-transition curve is not affected by motion speed /acceleration, but is determined only by the point location and the transitio n radius.



If the user sets the transition radius to be too large (exceeding the distance between the starting point/end point and the transition point), the system will automatically use half of the shorter distance between the starting point / end point and the transition point as the transition radius to calculate the transition curve.



When two consecutive transition radii (r1 and r2 in the figure below) coincide, the system will take the point after the first motion transition is completed as the starting point of the second motion, and then calculate the second r that actually takes effect according to th e logic of the transition radius being too large.



default value

is not specified via optional parameters, the smooth transition ratio set in the motion parameters will be used by default (see CP command for details ; the default value is 0 when the command is not called).

Precautions

A smooth transition will cause the robotic arm to move without passing through the midp oint. Therefore, when a smooth transition is set, the IO signal output or function setting (such as switching on and off the safety skin) between two motion commands **will be ex ecuted during the transition** .

If you want the command to be executed when the robotic arm reaches the midpoint precisel y, set the smooth transition parameter of the previous command to 0.

Because cp and r are implemented differently, if other instructions that do not affect the motion (such as conditional judgment, I/O, or function setting) are inserted between two motion instructions that require a smooth transition , cp and r will also handle them differently.

- If the **c p** method is used for transition, **most instructions will not affect the tran sition** , unless the processing time of the instruction is too long (such as the Wai t instruction).

In the example below , inserting an if statement between two motion instructions will not affe ct the smooth transition of the cp method.

```
-- The transition can proceed normally. The robot will check DI1 when it is about to reach point P1 . If it is ON , it will transition to the next motion instruction with a smooth transition ratio of 50% .
Mo vL(P1,{cp= 50 })
if (DI( 1 )==ON)  th
en
    Mo vL(P2)
en d
```

- If using the **'r'** method for transition, **only the whitelisted instructions below will not af fect the smooth transition** . Inserting any other instructions will disable the smooth tra nsition using the 'r' method .

```
RelPointTool, RelPointUser, DOGroup, DO, AO, ToolDO,
SetUser, SetTool, User, Tool, CP, AccJ, AccL, VelL, VelJ,
SetPayload, SetCollisionLevel, SetBackDistance, EnableSafeSkin, SetSafeSkin
```

In the example below , inserting an if statement between two motion instructions will cause t he r-mode smooth transition setting to fail

```
-- The smooth transition parameter is invalid, and the robot will determine DI1 after reaching point P1 .
MovL(P 1,{r= 5 })
if  (DI( 1 )==ON)
th en
    Mo vL(P2)
en d
```

Stop condition

The optional parameter `stopcond` specifies the motion-stopping condition; this parameter is a string expression. If a stop condition is specified... Stop conditions : During the execution of motion commands, the robot will check the stop conditions in real time, and end the current motion when the stop conditions are met, and directly execute the next command.

The stopping condition supports any expression that conforms to Lua syntax. The condition is considered satisfied when the expression is true. Typical applications are as follows:

```
-- Stop moving when DI1 is ON
MovJ(P1,{stopcond= "DI(1) == ON" })
-- The motion stops when both DI1 and DO1 are ON .
MovJ(P1,{stopcond= "DI(1) == ON and GetDO(1) == ON" })
-- Stop moving when the value stored at register address 100 is less
than 1 .          MovJ(P1,{stopcond= "GetHoldRegs(id, 100, 1)[1] <
10" })
-- The motion stops when variable var1 is not equal to 5 .
MovJ(P1,{stopcond= "var1 ~= 5" })
```

illustrate :

- Specifying a stopping condition will invalidate the smooth transition parameter r.
- specifying a stop condition, the smooth transition parameter cp will take effect, but the transition segment (curved portion) is not within the scope of the stop c
- ondition .
  If a smooth transition is added to the instruction preceding the stop condition, th e stop condition will not be triggered until the instruction leaves the transition s egment.

I/ O signal representation method

The system internally predefines ON and OFF variables to indicate whether there is a signal fo r digital I/O.

- ON = 1 indicates that a signal is present.
- OFF = 0 indicates no signal.

The ON|OFF parameter indicates that the parameter is ON or OFF, and the user can also use 1 or 0 as input.

Movement instructions

Instruction List

Motion commands are used to control the movement of the robotic arm. **Please read** the general instructions before use .

instruction	Function
M ovJ	Joint movement
M ovL	linear motion
A rc	Circular interpolation motion
C ircle	Circular interpolation motion
M ovJIO	Joint movement and output DO
M ovLIO	Linear motion and output DO
Ge tPathStartPose	Obtain the starting point of the trajectory
StarPath	Reproduce the recorded motion trajectory
PositiveKin	The joint angle is correctly interpreted as the position.
In verseKin	Inversely converting pose into joint angles

M ovJ

prototype :

```
MovJ(point, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = "expression"})
```

describe :

from the current position to the target point using joint movements.

Required parameters:

po int: target point.

Optional parameters:

- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.

- a: The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- cp: Smooth transition ratio. Value range: [0, 100].
- `stopcond`: A stop condition expression. When this condition is met, the current motion will end and the next instruction will be executed. For detailed information, please refer to the general instructions .

Example :

```
-- The robotic arm moves its joints to point P1 with the default settings .
Mo vJ(P1)
```

```
-- The robotic arm moves its joints to the specified joint angle using the default settings.
MovJ({joint={ 0 , 0 , 90 , 0 , 90 , 0 } })
```

```
-- The robotic arm joints move to the designated pose, corresponding to both user coordinate system 1 and tool coordinate system 1. Both acceleration and velocity are 50% , with a smooth transition ratio of 5. 0% .
MovJ({pose={ 300 , 200 , 300 , 180 , 0 , 0 } },{user= 1 ,tool= 1 ,a= 50 ,v= 50 ,cp= 50 })
```

```
-- Define the point first, then call it in the motion command; the running effect is the same as the previous command.
customPoint={pose={300,200,300,180,0,0}    }
MovJ(customPoint,{user=1,tool=1,a=50,v=50,cp=50})
```

```
-- The robotic arm joint moves to point P1 . The current movement ends
when DI1 is ON during the movement . MovJ(P1,{stopcond= "DI(1) == ON"
})
```

M ovL

prototype :

```
MovL(point, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "exp ression"})
```

describe :

Move from the current position to the target point in a straight line.

Required parameters:

po int: target point.

Optional parameters:

- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.
- a: The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

  setting this parameter, the parameter v will be ignored.
- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100],

  unit: mm. Setting this parameter will ignore the

  parameter cp.
- `stopcond`: A stop condition expression. When this condition is met, the

current motion will end and the next instruction will be executed. For detailed

information, please refer to the general instructions .

Example :

```
-- The robotic arm moves in a straight line to point P1 using the default settings .
Mo vL(P1)
```

```
-- The robotic arm moves linearly to point P1 at an absolute speed of 500 m/s .
Mo vL(P1,{speed= 500 })
```

```
-- The robotic arm moves linearly to the specified joint angle using the default settings.
MovL({joint={ 0 , 0 , 90 , 0 , 90 , 0 } })
```

```
-- The robotic arm moves linearly to the designated pose, corresponding to both user coordinate system 1 and tool coordinate system 1. Both acceleration and velocity are 50% , and the smooth transition radius is 5. mm .
MovL({pose={ 300 , 200 , 300 , 180 , 0 , 0 } },{user= 1 ,tool= 1 ,a= 50 ,v= 50 ,r= 5 })
```

```
-- Define the point first, then call it in the motion command; the running effect is the same as the previous command.
customPoint={pose={ 300 , 200 , 300 , 180 , 0 , 0 }   }          Mo
vL(customPoint,{user= 1 ,tool= 1 ,a= 50 ,v= 50 ,r= 5 })
```

```
-- When both speed and v are carried , speed takes effect, and the controller
will print the corresponding warning log.
-- If both cp and r are included , r will take effect, and the controller will print the
corresponding warning log.
MovL(P1,{v= 50 ,speed= 500 ,cp= 60 ,r= 5 })   --Optional parameters during execution: only speed and r take effect.
```

```
-- The robotic arm moves in a straight line to point P1 . The current
movement ends when DI1 is ON . MovL(P1,{stopcond= "DI(1) == ON" })
```

A rc

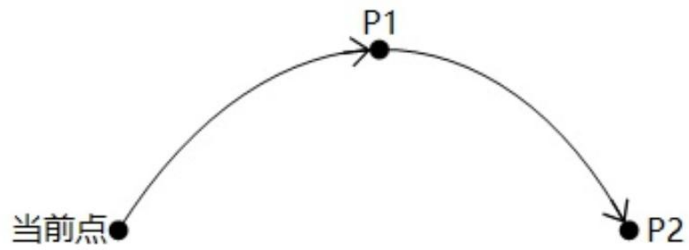
prototype :

```
Arc(P1, P2, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "exp ression"})
```

describe :

from the current position to the target point using circular interpolation.

needs to be determined using the current position, points P1, and P2. Therefore, the current p osition cannot be on the straight line determined by P1 and P2.



the movement is calculated by interpolating the postures of the current point and point P 2. The posture of point P1 is not included in the calculation (that is, the posture of the rob otic arm when it reaches point P1 during the movement may be different from the taught posture).

Required parameters:

- P1: Midpoint of the arc.
- P2: Target point.

Optional parameters:

- user : The user coordinate system of the target point.
- to ol : The tool coordinate system of the target point.
- a : The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

  setting this parameter, the parameter v will be ignored.
- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100],

  unit: mm. Setting this parameter will ignore the

  parameter cp.
- `stopcond`: A stop condition expression. When this condition is met, the

current motion will end and the next instruction will be executed. For detailed information, please refer to the general instructions .

Example :

```
-- The robotic arm moves to P1 , and then moves to P3 via P2 in a circular motion with the default settings .
Mo vJ(P1)
Ar c(P2,P3)
```

```
- - The robotic arm moves to P1 , and then moves to P3 via the arc of point {300,200,300,180,0,0}. The user coordinate system and the tool coordinate system are both 1 , and the motion acceleration and velocity are both 50% .
Mo vJ(P1)
Arc({pose={ 300 , 200 , 300 , 180 , 0 , 0 } },P3,{user= 1 ,tool= 1 ,a= 50 ,v= 50 })
```

```
-- The robotic arm moves to P1 , and then moves to P3 via P2 in a circular motion with the default settings . The current movement will end when DI1 is ON during the movement .
I vJ(P1)
Arc(P2,P3,{stopcond= "DI(1) == ON" })
```

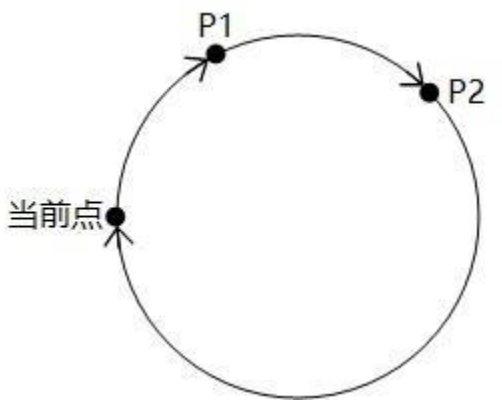
Circle

```
Circle(P1, P2, Count, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopc ond = "expression"})
```

describe :

from the current position, and return to the current position after a specified number of revolut ions.

needs to be defined using the current position, points P1, and P2. Therefore, the current position cannot be on the straight line defined by P1 and P2, and the complete circle defi ned by the three points cannot exceed the range of motion of the robotic arm.



the movement is calculated by interpolating the postures of the current point and point P 2. The posture of point P1 is not included in the calculation (that is, the posture of the rob otic arm when it reaches point P1 during the movement may be different from the taught posture).

Required parameters:

- P1: Full circle positioning point 1.
- P2: Full circle positioning point 2.
- Count: The number of full circular revolutions, ranging from [1, ...]. 999].

Optional parameters:

- user : The user coordinate system of the target point.
- to ol : The tool coordinate system of the target point.
- a : The percentage of the robotic arm's motion acceleration when this instruction is exec uted. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is execut ed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

setting this parameter, the parameter v will be ignored.

- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100], unit: mm. Setting this parameter will ignore the parameter cp.
- `stopcond`: A stop condition expression. When this condition is met, the current motion will end and the next instruction will be executed. For detailed information, please refer to the general instructions .

Example :

```
-- The robotic arm moves to P1 , and then performs a full circle along the path defined by P1 , P2 , and P3 .
Mo vJ(P1)
Ci rcle(P2,P3, 1 )
```

```
-- The robotic arm moves to P1 , and then moves 10 full circles along P1 , points {300,200,300,180,0,0} , and P3. The user coo
rdinate system and the tool coordinate system are both set to 1 , and the motion acceleration and velocity are both 50% .
Mo vJ(P1)
Circle({pose={300,200,300,180,0,0} },P3,10,{user=1,tool=1,a=50,v=50})
```

```
-- The robotic arm moves to P1 , and then moves around the circle defined by P1 , P2 , and P3 . The current movement ends when DI1 is ON
during the movement.
Mo vJ(P1)
Circle(P2,P3, 1 ,{stopcond= "DI(1) == ON" })
```

M ovJIO

prototype :

```
MovJIO(P,{    {Mode,Dis tance,Index,Status},{Mode,Distance,Index,Status} ... }, {user = 1 , tool = 0 , a = 20 , v = 50 , cp = 100 })
```

describe :

from the current position to the target point using joint motion, and set the status of the digita l output port in parallel during the movement.

Required parameters:

- P : Target point.
- Parallel digital output parameters: Sets the trigger point (DO) when the robotic arm moves a specified distance or percentage. Multiple groups can be set, each containing the following parameters:
  - Mode: Trigger mode. 0 indicates percentage triggering, 1 indicates distance triggering. The system will synthesize the angles of each joint into an angle vector and calculate the angle difference between the starting and ending points as the total distance of the movement.
  - Distance : Specifies a percentage/angle. Since angle calculations use composite angle vectors, it is recommended to use percentage mode for a more intuitive result .
    - When Distance is 0, it means that the trigger is initiated at the starting point.
    - When Distance is a positive number, it represents the percentage/angle of distance from the starting point.
    - When Distance is negative, it represents the percentage/angle of distance from the target point.
    - When Mode is 0, Distance represents the percentage of the total angle. Value range: (0, 100).
    - When Mode is set to 1, Distance represents the value of the angle. Unit: °.
  - Index: The number of the DO terminal.
  - Status: The DO state to be set. 0 and OFF indicate no signal, and 1 and ON indicate a signal.

Optional parameters:

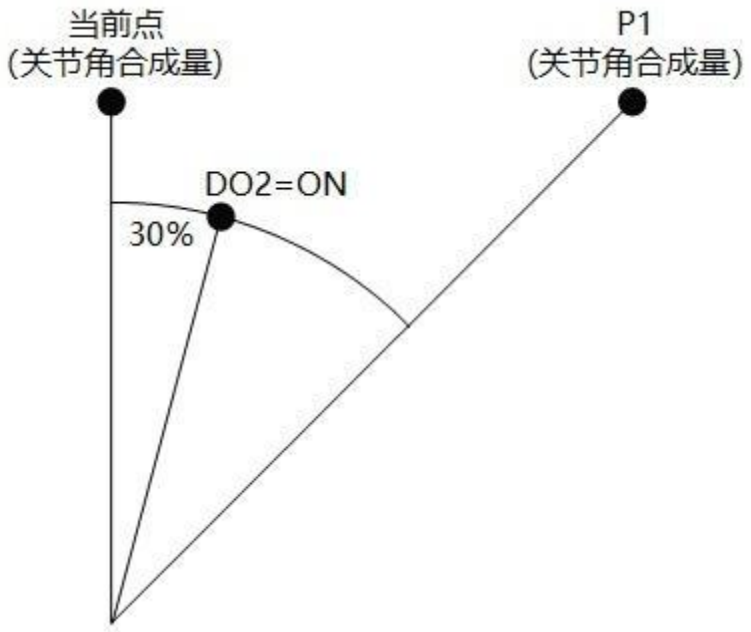
- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.
- a: The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- cp: Smooth transition ratio. Value range: [0, 100].

Smooth transitions can alter the robotic arm's trajectory and affect the timing of DO outputs, so please use them with caution.

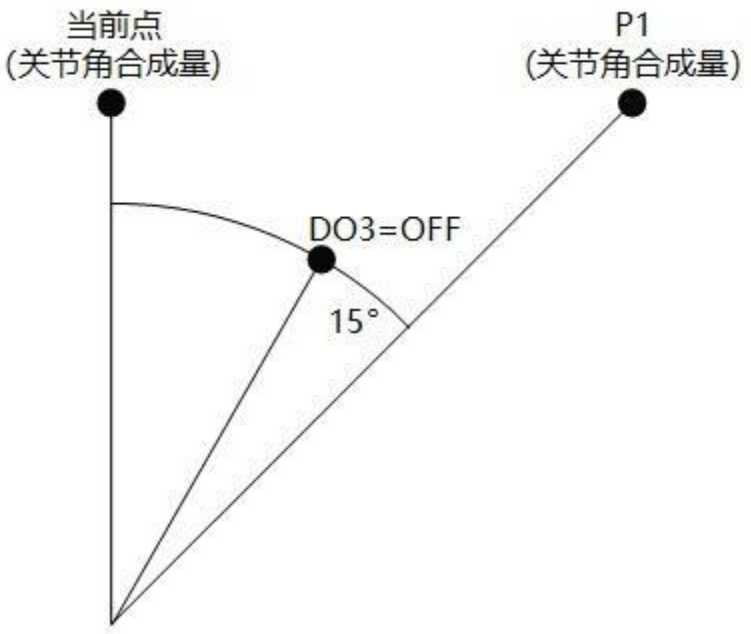
For detailed information, please refer to the General Instructions .

Example :

```
-- The robotic arm moves towards joint P1 with default settings . When it reaches a position 30% away from the starting point , DO2 is set to "on".
MovJIO(P1, { { 0 , 30 , 2 , 1 } })
```



-- The robotic arm moves towards joint P1 with default settings . When it reaches a position 15° away from the endpoint , DO3 is turned off.
MovJIO(P1, { { 1 , -15 , 3 , 0 } })



-- The robotic arm moves towards joint P1 with default settings . When it moves to a position 30% away from the starting p oint , DO2 is set to "on". When it moves to a position 20% away from the ending point , DO2 is set to "off".
MovJIO(P1, { { 0 , 30 , 2 , 1 } , { 0 , -20 , 2 , 0 } })

M ovLIO

prototype :

```
MovLIO(P,{    {Mode,Dis tance,Index,Status},{Mode,Distance,Index,Status} ... },{user = 1 , tool = 0 , a = 20 , v = 50 , speed = 500 , cp = 100 , r = 5 })
```

describe :

Move from the current position to the target point in a straight line, and set the status of the di gital output port in parallel during the movement.

Required parameters:

- P : Target point.
- Parallel digital output parameters: Sets the trigger point (DO) when the robotic arm moves a specified distance or percentage. Multiple groups can be set, each containing the following parameters:
  - Mode: Trigger mode. 0 indicates percentage-based triggering, and 1 indicates distan ce-based triggering.
  - Distance: Specifies a percentage/distance.
    - When Distance is 0, it means that the trigger is initiated at the starting point.
    - When Distance is a positive number, it represents the percentage/distance from t he starting point.
    - When Distance is negative, it represents the percentage/distance from the target point.
    - When Mode is 0, Distance represents the percentage of the total distance. Value range: (0, 100).
    - When Mode is set to 1, Distance represents the distance value. Unit: mm.
  - Index: The number of the DO terminal.
  - Status: The DO state to be set. 0 and OFF indicate no signal, and 1 and ON indicate a signal.

Optional parameters:

- user : The user coordinate system of the target point.
- to ol : The tool coordinate system of the target point.
- a : The percentage of the robotic arm's motion acceleration when this instruction is exec uted. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is execut ed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

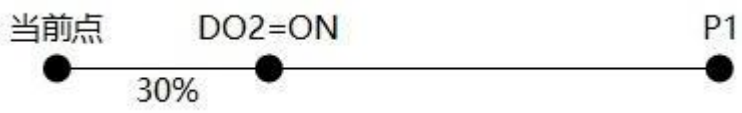
Setting this parameter will cause parameter v to be ignored.

- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100], unit: mm. Setting this parameter will ignore the parameter cp.

Smooth transitions can alter the robotic arm's trajectory and affect the timing of DO outputs; please use them with caution. For detailed instructions, please refer to the General Instructions .

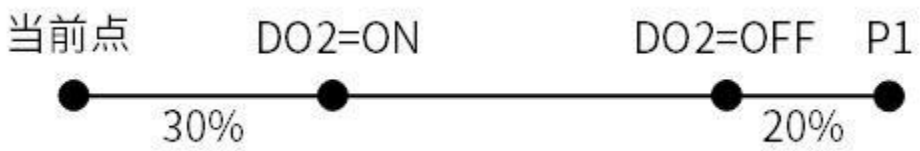
Example :

```
- - The robotic arm moves in a straight line towards point P1 with the default settings. When it reaches a position 30% away from the starting point , DO2 is set to "on".
MovL IO ( P1, { { 0 , 3 0 , 2 , 1 } })
```




```
-- The robotic arm moves in a straight line towards point P1 with the default settings. When it reaches a position 15mm away from the endpoint , DO3 is turned off.
MovLIO(P1, { { 1 , -15 , 3 , 0 } })
```


```
-- The robotic arm moves in a straight line towards point P1 with the default settings . When it moves to a position 30% away from the starting point , DO2 is set to "on". When it moves to a position 20% away from the ending point , DO2 is set to "off".
MovLIO(P1, { { 0 , 30 , 2 , 1 } , { 0 , -20 , 2 , 0 } })
```



Ge tPathStartPose

prototype :

```
GetPathStartPose ( string )
```

describe :

generally used in conjunction with the StartPath command, which is invoked to move to t he starting point of the trajectory reproduction. Depending on the trajectory file, the type of point data (teach point/joint/pose) may also vary.

Required parameters:

stri ng: Track file name (including file extension).

Example :

```
-- Get the starting point of the track.csv track file and print it.
local StartPoint = GetPathStartPose("track.csv") print(StartPoint)
-- StartPoint = {joint={j1,j2,j3,j4,j5,j6},"name" = "",user = userIndex,tool = toolIndex,pose
= {x,y,z,a,b,c}}
-- StartPoint = {joint={j1,j2,j3,j4,j5,j6}}
-- StartPoint = {pose = {x,y,z,a,b,c}}
```

StarPath

prototype :

```
StartPath(string, {multi = 1, isConst = 0, sample = 50 , freq = 0.2, user = 0,tool = 0})
```

描述:

Reproduce the trajectory recorded in the specified trajectory file. Before calling this command, the user needs to manually move the robotic arm to the starting point of the trajectory.

Required parameters:

stri ng: Track file name (including file extension).

Optional parameters:

- multi: The speed multiplier for reproduction, only valid when isConst=0. Value range: [0.1, ... 2], the default value is 1 when no parameter is specified .
- `is Const`: Whether to reproduce at a constant speed. The default value is 0 when no pa rameter is specified.
  - 1 indicates uniform speed reproduction; the robotic arm will reproduce the trajectory at a uniform speed according to the global rate.

    - 0 indicates that the motion speed is reproduced at the original speed when the trajectory was recorded, and the motion speed can be scaled proportionally using the multi parameter. In this case, the motion speed of the robotic arm is not affected by the global speed.
- sample : The sampling interval of trajectory points, i.e., the sampling time difference between two adjacent points when generating the trajectory file. Value range: [8, 1000], unit: ms, the default value is 50 when no parameter is specified (the sampling interval when the controller records the trajectory file).
- freq: Filter coefficient. The smaller the value of this parameter, the smoother the reproduced trajectory curve, but the more severe the distortion relative to the original trajectory. Please set an appropriate filter coefficient according to the smoothness of the original trajectory . Value range: (0,1], 1 indicates filtering is off, no parameter is specified. The default value is 0.2.
- user: Specifies the user coordinate system index corresponding to the trajectory poi nt. If not specified, the user coordinate system index recorded in the trajectory file i s used .
- tool: Specifies the tool coordinate system index corresponding to the trajectory poin t. If not specified, the tool coordinate system index recorded in the trajectory file is used .

Example :

```
-- track.csv is a file used by users through CBSH Studio Track files recorded by Pro .
-- After the joint moves to the starting point of the trajectory file, the trajectory
recorded in the file is reproduced at twice the original speed .  StartPoint  =
GetPathStartPose( "track.csv" )                    Mo vJ(StartPoint)
StartPath( "track.csv" , {multi = 2 , isConst = 0 })
```

```
--  track.csv is a file used by users through CBSH Studio .  Track files
recorded by Pro .
-- After the joint moves to the starting point of the trajectory file, it
reproduces the trajectory recorded in the file at a constant speed .
StartPoint  =  GetPathStartPose( "track.csv" )          Mo vJ(StartPoint)
StartPath( "track.csv" , {isConst = 1 })
-- customtrack.csv is a track file generated by the user, with a sampling interval of 20ms .
-- After the joint moves to the starting point of the trajectory file, the trajectory recorded in the file is reproduced at a consta
nt speed with a filter coefficient of 1 (to completely restore the recorded trajectory). The user and tool coordinate systems are b
oth 0 .
local StartPoint = GetPathStartPose( "customtrack.csv" ) Mo vJ(StartPoint)
StartPath("customtrack.csv", {isConst = 1, sample = 20, freq = 1, user = 0, tool = 0})
```

PositiveKin

```
PositiveKin(joint, {user = 1, tool = 0})
```

Perform forward calculus: Given the angles of each joint of the robotic arm, calculate the coo rdinates of the robotic arm's end effector in the given Cartesian coordinate system.

Required parameters

joint: Joint variable, in the format `{joint = {j1, j2, j3, j4, j5, j6} }`

Optional parameters

- user: User coordinate system index. If not specified, the global user coordinate system is used.
- tool: Tool coordinate system index. If not specified, the global tool coordinate system is us ed.

return

The pose variables obtained from the correct solution are in the format `{pose = {x, y, z, rx, ry, rz}. }`

Example

```
-- Given the joint coordinates as {0,0,-90,0,90,0} , calculate the Cartesian coordinates of the robotic arm's end effector in the global user / to ol coordinate system.
PositiveKin({joint={ 0 , 0 , -90 , 0 , 90 , 0 } })
```

```
-- Given the joint coordinates as {0,0,-90,0,90,0} , calculate the Cartesian coordinates of the robotic arm's end effector in user coordinate sys tem 1 and tool coordinate system 1 .
PositiveKin({joint={0,0,-90,0,90,0} },{user=1,tool=1})
```

InverseKin

```
I nverse Kin(pose, { useJo intNear = true , j oi ntNear = j oint, user = 1 , tool = 0 })
```

describe

Perform inverse kinematics: Given the coordinates of the robotic arm's end effector in a give n Cartesian coordinate system, calculate the angles of each joint of the robotic arm.

Since Cartesian coordinates only define the spatial coordinates and tilt angle of the inver se joint (TCP) , the robotic arm can reach the same pose through various different postur es , meaning that one pose variable can correspond to multiple joint variables. To obtain a unique solution, the system needs a specified joint coordinate and selects the solution

closest to that joint coordinate as the inverse solution result.

Users can use this command to confirm whether a certain Cartesian point is reachable.

Required parameters

pose: A pose variable, in the format `{pose = {x, y, z, rx, ry, rz} }`

Optional parameters

- useJointNear: A boolean value used to determine whether the JointNear parameter is valid.
  - true indicates that the nearest solution is selected based on the JointNear parameter.
  - False or no parameter indicates that the JointNear parameter is invalid, and the system selects the nearest solution based on the current joint angle of the robotic arm.
  - This parameter is invalid if it is only included without the jointNear parameter.
- jointNear：用于就近选解的关节变量，格式为 `{joint = {j1, j2, j3, j4, j5, j6} }`
- user: User coordinate system index. If not specified, the global user coordinate system is used.
- tool: Tool coordinate system index. If not specified, the global tool coordinate system is used.

return

- Error codes: 0 indicates successful inversion, -1 indicates failed inversion (no solution).
- The joint variables obtained from the inverse solution are in the format `{joint = {j1, j2, j3, j4, j5, j6}` When the inverse solution fails, all values from j1 to j6 `are` 0 .

Example

```
--The Cartesian coordinates of the robotic arm's end effector in the global user / tool coordinate system are {300, 200, 300, 180, 0, } 0} , calculate the joint coordinates and choose the nearest solution.

local errId, jointPoint = InverseKin({ pose = { 300 , 200 , 300 , 180 , 0 , 0 } })
```

```
-- The Cartesian coordinates of the robotic arm's end effector in user coordinate system 1 and tool coordinate system 1 are { 300, } 200, 300, 180, 0, 0} , calculate the joint coordinates and choose the nearest solution.
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} }, {user=1, tool=1})
```

```
-- The Cartesian coordinates of the robotic arm's end effector in the global user / tool coordinate system are {300, } 200, 300, 180, 0, 0} , calculate joint coordinates, selecting the distance from the joint angle {90, 30, -90, 180, 30, }. The nearest solution is 0 .
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} }, {useJointNear = true, jointNear = { joint = {90, 30, -90, 180, 30, 0} } })
```

Relative motion command

Instruction List

The relative motion command function is used to control the robotic arm to perform offset mo vements. **Please read** the general instructions before use .

instruction	Function
Re lPointUser	Point offset along the user coordinate system
Re lPointTool	Point offset along the tool coordinate system
Re lMovJTool	relative joint motion along the tool coordinate system
Rel MovLTool	Relative linear motion along the tool coordinate system
Re lMovJUser	relative joint motion along the user coordinate system
Rel MovLUser	Relative linear motion along the user coordinate system
Rel JointMovJ	Joint movement to a specified offset angle
Re lJoint	Point offset specifies joint angle

RelPointUser

prototype :

```
RelPoin tUser (P, { Of fsetX, O f fsetY, O f fsetZ, Of setRx , Of setRy , O ffs etRz})
```

describe :

a specified point along a specified user coordinate system and return the offset point.

Required parameters:

- P : The specified point to be offset.
- {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, `OffsetRz}`: Specifies the offset in the user coordinate system. x , y, z represent spatial offsets in mm; rx, ry, rz represent angular offsets in °.
  - If the specified point is a teaching point, the offset is performed based on the user co ordinate system of the teaching point.
  - If the specified point is a joint variable or pose variable, the offset is based on the glo bal user coordinate system .

return :

- If the specified point is a teaching point, return the teaching point constant after offset.
- If the specified point is a joint variable or pose variable, return the offset pose variable: {p ose = {x, y, z, rx, ry, rz}}.

Example :

```
-- Offset P1 by a certain distance in the specified user coordinate system, and then move it to the offset point.
local Offset={OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}
local p = RelPointUser(P1, Offset) Mo vL(p)
```

RelPointTool

prototype :

```
RelPoin tTool (P, { Of fsetX, O f fsetY, O f fsetZ, Of setRx , Of setRy , O ffs etRz})
```

describe :

a specified point along the specified tool coordinate system and return the offset point.

Required parameters:

- P : The specified point to be offset.
- {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, `OffsetRz}`: Specifies the offset in the tool's coordinate system. `x` , `y`, and `z` represent spatial offsets in mm; `rx`, `ry`, and `rz` represent angular offsets in degrees.
  - If the specified point is a teaching point, the offset is performed based on the tool co ordinate system of the teaching point.
  - If the specified point is a joint variable or pose variable, the offset is based on the glo bal tool coordinate system .

return :

- If the specified point is a teaching point, return the teaching point constant after offset.
- If the specified point is a joint variable or pose variable, return the offset pose variable: {p ose = {x, y, z, rx, ry, rz}}.

Example :

```
-- Offset P1 by a certain distance in the specified tool coordinate system, and then move it to the offset point.
local Offset={OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}
local p = RelPointTool(P1, Offset) Mo vL(p)
```

RelMovJTool

prototype :

```
RelMovJTool({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = " expression"})
```

describe :

from its current position along the specified tool coordinate system using joint motion. Th e trajectory of the joint motion is not linear, and all joints move simultaneously .

Required parameters:

{x, y, z, rx, ry, rz}: The offset of the target point relative to the current position in the spe cified tool coordinate system. x, y, z represent spatial offsets in mm; rx, ry, rz represent angular offsets in °.

Optional parameters:

- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.
- a: The percentage of the robotic arm's motion acceleration when this instruction is execut ed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- cp: Smooth transition ratio. Value range: [0, 100].
- `stopcond`: A stop condition expression. When this condition is met, the current motion will end and the next instruction will be executed. For detailed information, please refer to the general instructions .

Example :

```
-- The robotic arm moves along the global tool coordinate system joints to the specified offset point using the default settings.
RelMovJTool({ 10 , 10 , 10 , 0 , 0 , 0 })
```

For more examples of optional parameters, please refer to MovJ.

RelMovLTool

prototype :

```
RelMovLTool({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "expression"})
```

describe :

from the current position along the specified tool coordinate system.

Required parameters:

{x, y, z, rx, ry, rz}: The offset of the target point relative to the current position in the spe cified tool coordinate system. x, y, z represent spatial offsets in mm; rx, ry, rz represent angular offsets in °.

Optional parameters:

- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.
- a: The percentage of the robotic arm's motion acceleration when this instruction is execut ed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

  setting this parameter, the parameter v will be ignored.
- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100],

  unit: mm. Setting this parameter will ignore the

  parameter cp.
- `stopcond`: A stop condition expression. When this condition is met, the

current motion will end and the next instruction will be executed. For detailed

information, please refer to the general instructions .

Example :

```
-- The robotic arm moves in a straight line along the global tool coordinate system to the specified offset point using the default settings.
RelMovLTool({ 10 , 10 , 10 , 0 , 0 , 0 })
```

For more examples of optional parameters, please refer to MovL.

RelMovJUser

prototype :

```
RelMovJUser({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = " expression"})
```

describe :

starting position along the specified user coordinate system using joint motion. The traject ory of the joint motion is not linear, and all joints move simultaneously .

Required parameters:

{x, y, z, rx, ry, rz}: The offset of the target point relative to the current position in the spe cified user coordinate system. x, y, z represent spatial offsets in mm; rx, ry, rz represent angular offsets in °.

Optional parameters:

- user: The user's coordinate system for the target point.
- tool: The tool coordinate system of the target point.
- a: The percentage of the robotic arm's motion acceleration when this instruction is execut ed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- cp: Smooth transition ratio. Value range: [0, 100].
- `stopcond`: A stop condition expression. When this condition is met, the current motion will end and the next instruction will be executed. For detailed information, please refer to the general instructions .

Example :

```
-- The robotic arm moves along the global user coordinate system joints to the specified offset point using the default settings.
RelMovJUser({ 10 , 10 , 10 , 0 , 0 , 0 })
```

For more examples of optional parameters, please refer to MovJ.

RelMovLUser

prototype :

```
RelMovLUser({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "expression"})
```

describe :

from the current position along the specified user coordinate system in a linear motion.

Required parameters: {x, y, z, rx, ry, rz}: The offset of the target point relative to the current position in the specified user coordinate system. x, y, z represent spatial offsets in mm; rx, ry, rz represent angular offsets in °.

Optional parameters:

- user : The user coordinate system of the target point.
- to ol : The tool coordinate system of the target point.
- a : The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

  setting this parameter, the parameter v will be ignored.
- c p: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100],

  unit: mm. Setting this parameter will ignore the

  parameter cp.
- `stopcond`: A stop condition expression. When this condition is met, the

current motion will end and the next instruction will be executed. For detailed

information, please refer to the general instructions .

Example :

```
-- The robotic arm moves in a straight line along the global user coordinate system to the specified offset point using the default settings.
RelMovLUser({ 10 , 10 , 10 , 0 , 0 , 0 })
```

For more examples of optional parameters, please refer to MovL.

RelJointMovJ

prototype :

```
RelJointMovJ({Offset1, Offset2, Offset3, Offset4, Offset5, Offset6}, {a = 20 , v = 50 , cp = 100 })
```

describe :

from the current position to the specified joint offset angle using joint motion.

Required parameters:

{Offset1, Offset2, Offset3, Offset4, Offset5, Offset6}: J1 axis in joint coordinate system ~ Offset value along the J6 axis , in degrees.

Optional parameters:

- a: The percentage of the robotic arm's motion acceleration when this instruction is executed. Value range: (0, 100).
- v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
- cp: Smooth transition ratio. Value range: [0, 100]. Please refer to the general instructions for details .

Example :

```
-- The robotic arm moves its joints to the offset angle with the default settings.
Re lJointMovJ({ 20 , 20 , 10 , 0 , 10 , 0 })
```

R elJoint

prototype :

```
RelJoint(P,  {Offset1, Offset2 , Offset3 ,  Offset4,  offset5, o ff set6})
```

describe :

to the specified point in the joint coordinate system, and return the joint variable after offset.

Required parameters:

- P: The specified point to be offset.
- Offset1~Offset6: J1 axis in the joint coordinate system Offset value along the J6 axis, in degrees.

return :

Offset joint variables: {joint = {j1, j2, j3, j4, j5, j6}}.

Example :

```
-- Offset P1 by a certain angle on axes J1 to J6 respectively, and then move it to the offset position.
local Offset = {Offset1, Offset2, Offset3, Offset4, offset5, offset6} local p = RelJoint(P1, Offset)
Mo vJ(p)
```

Motion parameters

Instruction List

Motion parameter functions are used to set or retrieve motion-related parameters of the robot ic arm. **Please read** the general instructions before use .

instruction	Function
C P	Set the smooth transition ratio
V elJ	Set the speed ratio of joint movement modes
A ccJ	Set the acceleration ratio for joint movement patterns .
V elL	Set the speed ratio for linear and arc motion modes.
A ccL	Set the acceleration ratio for linear and arc motion.
SpeedFactor	Set the global motion rate of the robotic arm
Se tPayload	Configure end load
U ser	Set global user coordinate system
SetUser	Modify the specified user coordinate system
Ca lcUser	Calculate the user coordinate system
To ol	Set global tool coordinate system
SetTool	Modify the specified tool coordinate system
Ca lcTool	Calculation tool coordinate system
G etPose	Obtain the real-time pose of the robotic arm
Ge tAngle	Obtain the real-time joint angles of the robotic arm
G e tABZ	Get the current position of the encoder
Ch eckMovJ	Check the operability of joint movement commands.
Ch eckMovL	Check the feasibility of linear or arc motion commands.
Set SafeWallEnable	Enable or disable a specified security wall
Set WorkZoneEnable	Enable or disable a specified security zone
S etC ollisionLevel	Set collision level
Set BackDistance	Set collision backoff distance

C P

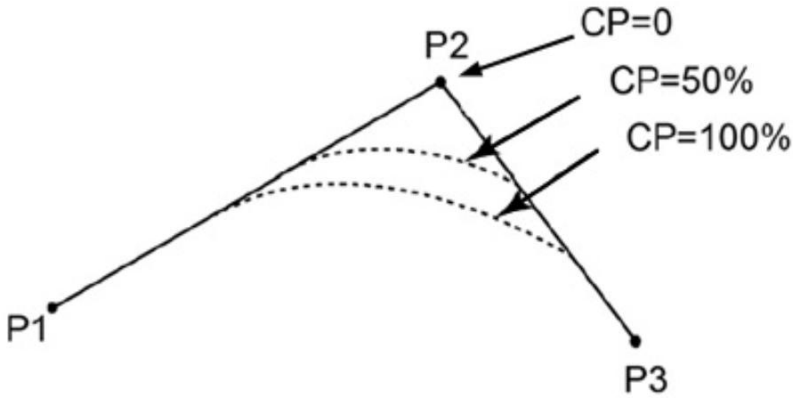
prototype :

```
C P(R)
```

describe :

Set the smooth transition ratio, that is, when the robotic arm moves continuously through multiple points, whether the transition at the intermediate point is a right angle or a curve .

this command only applies to the current project run; the default value is 0 if not set.



Required parameters:

R: Smooth transition ratio. Value range: [0, ... 100].

Example :

```
-- The robotic arm is currently at point P1 . It will smoothly transition at a 50% transition ratio before reaching point P2 , and finally reach p
oint P3 .
C P( 50 )
Mo vL(P2)
Mo vL(P3)
```

V elJ

prototype :

```
V e lJ(R)
```

describe :

sets the speed ratio for the joint motion mode

(MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ). The speed ratio set by this

command only applies to the current project running; the default value is 100 if not set.

Required parameters:

R: Speed ratio. Value range: [1, 100].

Example :

```
-- The robotic arm moves to point P1 at a speed of 20% .
Ve lJ( 20 )
Mo vJ(P1)
```

A ccJ

prototype :

```
A c cJ(R)
```

describe :

Set the acceleration ratio for the joint motion mode (MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ) .

this command only takes effect during the current project run; the default value is 100 when not set.

Required parameters:

R: Acceleration ratio. Value range: [1, 100].

Example :

```
-- The robotic arm moves to point P1 with a 50% acceleration ratio .
Ac cJ( 50 )
Mo vJ(P1)
```

V elL

prototype :

```
V e lL(R)
```

describe :

Set the speed ratio for linear and arc motion modes (MovL/Arc/Circle/MovLIO/RelMovLTool/RelMovLUser) .

this command only applies to the current project running; the default value is 100 when not set.

Required parameters:

R: Speed ratio. Value range: [1, 100].

Example :

```
-- The robotic arm moves to point P1 at a speed of 20% .
Ve lL( 20 )
Mo vL(P1)
```

A ccL

prototype :

```
A c cL(R)
```

describe :

Set the acceleration ratio for linear and arc motion modes (MovL/Arc/Circle/MovLIO/Rel MovLTool/RelMovLUser).

this command only takes effect during the current project run; the default value is 100 when not set.

Required parameters:

R: Acceleration ratio. Value range: [1, 100].

Example :

```
-- The robotic arm moves to point P1 with a 50% acceleration ratio .
Ac cL( 50 )
Mo vL(P1)
```

SpeedFactor

prototype :

```
S peedFact or (ratio)
```

describe :

To set the global motion rate of the robotic arm, please refer to the general instructions in the motion commands .

this command only takes effect during the current project run. If not set, the value set by t he control software (when running the script through the control software) or TCP comm and (when running the script through the TCP command) before running the script will be used.

Required parameters:

ratio : The ratio of motion speed. Value range: [1, 100].

Example :

```
- - Set the global motion rate to 50% .
SpeedFactor ( 50 )
```

S etPayload

prototype :

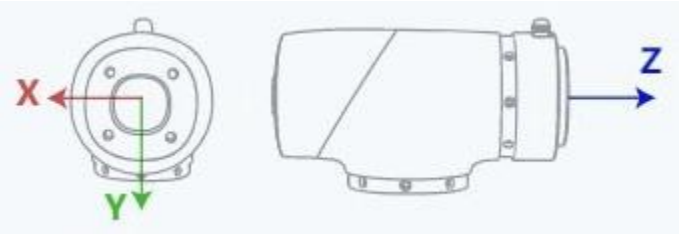
```
SetPayload(payload, {x, y, z}) Se
tPayload(name)
```

describe :

Sets the weight and eccentricity coordinates of the end load. It supports both direct settin g and setting via preset parameter groups. Parameters set by this command only take eff ect during the current project run, overriding previous settings; they revert to the original s ettings after the project stops.

Required parameter 1:

- Payload: The weight of the end load. Unit: kg.
- {x, y, z}: Eccentric coordinate of the end load, with the coordinate axis direction shown in the figure below. Unit: mm.



Example 1:

```
--Set the end load weight to 1.5kg , and the eccentric coordinates to {0, 0, } 10 }
SetPayload( 1.5 , { 0 , 0 , 10 })
```

Required parameter 2:

- name: The name of the preset load parameter group. The corresponding parameter group needs to be saved in the control software first.

Example 2:

```
-- Configure the end load using the preset load parameter
group named "load1" . ` SetPayload( "load1" )`
```

U ser

prototype :

```
User
```

describe :

Sets the global user coordinate system. If the user does not specify a user coordinate system via optional parameters when invoking other commands, the system will use the global user coordinate system.

this command only takes effect during the current project run. If not set, the default global user coordinate system is user coordinate system 0.

Required parameters:

user: The user coordinate system after switching, specified by index number. The corresponding coordinate system must first be added in the control software. If the specified coordinate system is empty or does not exist, the project will stop running and report an error.

Example :

```
-- Switch the global user coordinate system to user coordinate system 1 .
Use er( 1 )
-- The following example illustrates the effective range of the global user coordinate system.
MovL({pose={ 300 , 200 , 300 , 180 , 0 , 0 } }) -- Use the global user coordinate system ( 1 )
MovL({pose={ 300 , 200 , 300 , 180 , 0 , 0 } },{user= 2 }) --Use the user coordinate system specified by
the optional parameter ( 2 ) MovL ( P1) -- P1 is the teaching point, using the user coordinate system that
comes with the teaching point.
```

SetUser

prototype :

```
SetUser (index , table , type )
```

describe :

Modify the specified user coordinate system.

Required parameters:

- index: User coordinate system index. The corresponding coordinate system must be added in the control software first. If the specified coordinate system index does not exist, the project will stop running and report an error.
- table: The modified user coordinate system, formatted as {x, y, z, rx, ry, When dealing with rx, ry, and rz rotation values, it is recommended to use the CalcUser command to obtain them.

Optional parameters:

- type: Whether to save globally. The default value is 0.
  - 0 : The coordinate system modified by this command only takes effect during the current project run. After the project stops, it will revert to its original value.
  - 1 : This command saves the modified coordinate system globally, retaining the modified values even after the project stops. However, when `type=1` (all...) When saving the coordinate system , users need to manually switch to the corresponding coordinate system settings interface and then switch back to see the updated values , which means they need to manually refresh the interface.

Example :

```
-- Change the user coordinate system 1 to the new coordinate system calculated by CalcUser . This change only takes effect during project execution.
newUser = CalcUser( 1 , 1 ,{ 10 , 10 , 10 , 10 , 10 , 10 }) Se
tUser( 1 ,newUser)
User( 1 ) - - Switch the global user coordinate system to 1
```

```
-- Change the user coordinate system 1 to the new coordinate system calculated by CalcUser , and save it globally.
newUser = CalcUser( 1 , 1 ,{ 10 , 10 , 10 , 10 , 10 , 10 }) Se
tUser( 1 ,newUser, 1 )
MovL(P1,{user= 1 }) -- Use the optional parameter to specify the reference user coordinate system for this instruction as 1.
```

C alcUser

prototype :

```
C alcUser(in dex,matrix        direction,table)
```

describe :

Calculate the user coordinate system.

Required parameters:

- index: User coordinate system index, with a value range of [0,50]. User coordinate system 0 is initially set to the base coordinate system.
- matrix_direction : the direction of calculation.
  - 1: Left multiplication indicates that the coordinate system specified by index is deflected by the value specified by table along the base coordinate system.
  - 0: Right multiplication, indicating that the coordinate system specified by index is rotated along itself by the value specified by table.
- table: User coordinate system offset values, in the format {x, y, z, rx, ry, } rz}

return :

The calculated user coordinate system is in the format {x, y, ...}. z, rx, ry, rz}

Example :

```
-- The following calculation process can be equivalent to: taking a coordinate system with the same initial pose as user coordinate system 1 , and translating it along the base coordinate system {x=10,   y=10,   z=10} and rotate {rx=10, ry=10, After setting rz=10 , the new coordinate system obtained is newUser .
newUser = CalcUser(1,1,{10,10,10,10,10,10})
```

```
-- The following calculation process can be equivalent to: a coordinate system with an initial pose identical to user coordinate system 1 , translated along user coordinate system 1 by {x=10,  y=10,  z=10} and rotate {rx=10, ry=10, After setting rz=10 , the new coordinate system obtained is newUser .
newUser = CalcUser( 1 , 0 ,{ 10 , 10 , 10 , 10 , 10 , 10 })
```

Tool

prototype :

```
Tool
```

describe :

Switch the global tool coordinate system. If the user does not specify the tool coordinate system via optional parameters when invoking other commands, the system will use the global tool coordinate system.

this command only takes effect during the current project run. If not set, the default globa l tool coordinate system is tool coordinate system 0.

Required parameters:

`tool`: The new tool coordinate system, specified by index number. The corresponding co ordinate system must be added in the control software first. If the specified coordinate sy stem is empty or does not exist, the project will stop running and report an error.

Example :

```
-- Switch the global tool coordinate system to tool coordinate system 1 .
To ol( 1 )
-- The following example illustrates the scope of the global tool coordinate system.
MovL({pose={ 300 , 200 , 300 , 180 , 0 , 0 } }) -- Use the global tool coordinate system ( 1 )
MovL({pose={ 300 , 200 , 300 , 180 , 0 , 0 } },{tool= 2 }) --Use the tool coordinate system specified by the
optional parameter ( 2 ) MovL (P1) -- P1 is the teaching point, and the tool coordinate system provided by
the teaching point is used.
```

SetTool

prototype :

```
SetTool (index , table , type )
```

describe :

Modify the specified tool coordinate system.

Required parameters:

- index: The coordinate system index for the tool. The corresponding coordinate syste m must be added in the control software first. If the specified coordinate system ind ex does not exist, the project will stop running and report an error.
- table: The modified tool coordinate system, formatted as {x, y, z, rx, ry, When dealin g with rx, ry, and rz rotation values, it is recommended to use the CalcTool comman d to obtain them.

Optional parameters:

- type: Whether to save globally. The default value is 0.
  - 0 : The coordinate system modified by this command only takes effect during the cu rrent project run. After the project stops, it will revert to its original value.
  - 1 : This command saves the modified coordinate system globally, retaining the modified values even after the project stops. However, when `type=1` (all...) W h e n saving the coordinate system , users need to manually switch to the corresponding coordinate system settings interface and then switch back to see the updated values , which means they need to manually refresh the interface.

Example :

```
-- The tool coordinate system 1 has been changed to the new coordinate system calculated by CalcTool . This change only takes effect during project execution.
newTool = CalcTool( 1 , 1 ,{ 10 , 10 , 10 , 10 , 10 , 10 }) Se
tTool( 1 ,newTool)
Tool( 1 ) - - Switch the global tool coordinate system to 1
```

```
-- Change the tool's coordinate system 1 to the new coordinate system calculated by CalcTool , and apply it globally.
newTool = CalcTool( 1 , 1 ,{ 10 , 10 , 10 , 10 , 10 , 10 }) Se
tTool( 1 ,newTool, 1 )
MovL(P1,{tool= 1 }) -- Use the optional parameter to specify the reference tool coordinate system for this command as 1.
```

CalcTool

prototype :

```
C alcTool(in dex, matrix)      direction,table)
```

describe : Calculation tool coordinate system.

Required parameters:

- index: Tool coordinate system index, value range: [0,50], based on the initial value of coordinate system 0 as flange coordinate system (TCP0).
- matrix_direction : the direction of calculation.
  - 1: Left multiplication indicates that the coordinate system specified by index is deflected along the flange coordinate system (TCP0) by the value specified by table.
  - 0: Right multiplication, indicating that the coordinate system specified by index is rotated along itself by the value specified by table.
- table: Tool coordinate system, formatted as {x, y, z, rx, ry, } rz}

return :

The calculated tool coordinate system is in the format {x, y, ... z, rx, ry, rz}

Example :

```
-- The following calculation process can be equivalent to: a coordinate system with the same initial pose as tool coordinate system 1 , translated along the flange coordinate system ( TCP0 ) by {x=10,  y=10,  z
=10} and rotate {rx=10, ry=10, After setting rz=10 , the new coordinate system obtained is newTool .
newTool = CalcTool(1,1,{10,10,10,10,10,10})
```

```
-- The following calculation process can be equivalent to: Given a coordinate system with the same initial pose as tool coordinate system 1 , translate along tool coordinate system 1 by {x=10, }  y=10,  z=10} and rotate {rx=10, ry=10, After setting rz=10 , the new coordinate system obtained is newTool .
newTool = CalcTool( 1 , 0 ,{ 10 , 10 , 10 , 10 , 10 , 10 })
```

G etPose

prototype :

```
G etPo se(user     index,     t(oo l ndex)
```

describe :

Obtain the real-time pose of the robotic arm.

If this command is invoked between two motion commands, the acquired points will be affect ed by the smooth transition settings:

- If the smooth transition function is turned off (cp or r is 0), the target point can be accurat ely obtained.
- If the smooth transition function is enabled, a certain point on the transition curve will be o btained.

Optional parameters:

- use r_index : The user coordinate system index corresponding to the pose. The corr esponding coordinate system needs to be added in the control software first. If not set, the global user coordinate system is used .
- too l_index : The tool coordinate system index corresponding to the pose. The corres ponding coordinate system needs to be added in the control software first. If not set , the global tool coordinate system is used.

return :

Current pose of the robotic arm: {pose = {x, y, z, rx, ry, rz}}

Example :

```
-- The robotic arm first moves to P1 , and then returns to the current pose.
local currentPose = GetPose() Mo vJ(P1)
Mo vJ(currentPose)


-- The robotic arm moves to P1 and then to P2 to obtain the pose of P1 .
Mo vJ(P1,{cp= 0 })
local currentPose = GetPose() --Get the pose Mo vJ(P2) of
P1
```

```
-- Get the current pose in user coordinate system 1 and tool coordinate system 1 .
local currentPose = GetPose( 1 , 1 )
```

G etAngle

prototype :

```
G etA ngle()
```

describe :

Obtain the real-time joint angles of the robotic arm.

If this command is invoked between two motion commands, the obtained joint angles will be affected by the smooth transition settings:

- If the smooth transition function is turned off (cp or r is 0), the joint angle corresponding to the target point can be accurately obtained.
- If the smooth transition function is enabled, the joint angle corresponding to a certain point on the transition curve will be obtained.

return :

The current joint angle of the robotic arm: {joint = {j1, j2, j3, j4, j5, j6} }

Example :

```
-- The robotic arm first moves to P1 , and then returns to the current pose.
local currentAngle = GetAngle() Mo vJ(P1)
Mo vJ(currentAngle)



-- The robotic arm moves to P1 and then to P2 to obtain the joint angle of P1 .
Mo vJ(P1,{cp= 0 })
local currentAngle = GetAngle() --Get the joint angle Mo vJ(P2) of
P1
```

G etABZ

prototype :

```
G e tABZ()
```

describe :

Get the current position of the encoder.

Return value:
The encoder 's current position.

Example :

```
-- Get the current position of the set ABZ encoder and assign it to the variable abz .
local abz = GetABZ()
```

C h eckMovJ

prototype :

```
C heck MovJ(P, { user = 1 , t ool = 0 , a = 2 0 , v = 5 0 , cp = 1 00 })
```

describe :

Check the feasibility of joint motion from the current point to the target point. The system will calculate the entire motion trajectory and check for any unreachable points in the traje ctory .

Required parameters:

P : Target point.

Optional parameters:

user: The user's coordinate system for the target point.
tool: The tool coordinate system of the target point.
a: The percentage of the robotic arm's motion acceleration when this instruction is execute d. Value range: (0, 100).
v: The percentage of the robotic arm's movement speed when this instruction is executed. Value range: (0, 100).
cp: Smooth transition ratio. Value range: [0, 100]. Please refer to the general instructions for details .

return :

Inspection results.

0: No errors

16: The finish line is strangely close to the shoulder.

17: The inverse solution at the endpoint has no solution.

18: Endpoint Reverse Solution Limit

22: Gesture switching error

26: The end point is near the wrist, which is strange.

27: The finish line is strangely close to the elbow.

29: Speed parameter error

30: Failed to solve the inverse problem with all parameters

32: The trajectory has a shoulder anomaly.

33: The trajectory has an inverse solution but no solution point.

34: Trajectory contains inverse kinematic limit points

35: Trajectory contains wrist singularity points

36: Trajectory contains axis singularity points

37: Trajectory contains joint jump points

```
-- Check if P1 is reachable using default joint motion settings; if so, MoveJ to P1.
    local status=CheckMovJ(P1)
    if(status==0)
    then
          MovJ(P1)
status=CheckMovJ(P2)  -- Execute the feasibility check from P1 to P2 only after the robot arm reaches P1.
           if(status==0)
           then
                MovJ(P2)

       end
 end
```

CheckMovL

prototype :

```
CheckMovL(P, { user = 1 , t ool = 0 , a = 2 0 , v = 5 0 , cp = 1 00 , r = 5 })
```

describe :

Check the feasibility of linear motion from the current point to the target point. The system will calculate the entire trajectory and check for any unreachable points along the trajecto ry .

Required parameters:

P : Target point.

Optional parameters:

- user : The user coordinate system of the target point.
- to ol : The tool coordinate system of the target point.
- a : The percentage of the robotic arm's motion acceleration when this instruction is exec uted. Value range: (0, 100).
- v : The percentage of the robotic arm's movement speed when this instruction is execut ed. Value range: (0, 100).

- speed: The target speed of the robotic arm when this instruction is executed, with a value range of [1, ...]. [Maximum speed], unit: mm /s.

  setting this parameter, the parameter v will be ignored.
- cp: Smooth transition ratio. Value range: [0, 100].
- r: Smooth transition radius, value range: [0,100], unit: mm.

  Setting this parameter will

cause the cp parameter to be

ignored. For detailed instructions,

please refer to the general

documentation .

return :

Inspection results.

- 0: No errors
- 16: The finish line is strangely close to the shoulder.
- 17: The inverse solution at the endpoint has no solution.
- 18: Endpoint Reverse Solution Limit
- 22: Gesture switching error
- 26: The end point is near the wrist, which is strange.
- 27: The finish line is strangely close to the elbow.
- 29: Speed parameter error
- 30: Failed to solve the inverse problem with all parameters
- 32: The trajectory has a shoulder anomaly.
- 33: The trajectory has an inverse solution but no solution point.
- 34: The trajectory has inverse solution limit points.
- 35: The trajectory has a wrist singularity.
- 36: The trajectory has an axial singularity.
- 37: The trajectory contains joint jump points.

Example :

```
-- Check if P1 is reachable using the default linear motion settings . If it is reachable, then move linearly to P1 .
local status=CheckMovL(P1) if
(status== 0 )
th en
    Mo vL(P1)
    status=CheckMovL(P2)  --The feasibility check from P1 to P2 should be executed only after
    the robotic arm has moved to P1 , following the `if (status== 0 )` statement.
    then
        MovL(P2)
    end
en d
```

S et SafeWallEnable

prototype :

```
S etSafeWallEnable (index,value)
```

describe :

Enables or disables the specified security wall. The security wall settings configured by thi s command only take effect during the current project run; they revert to their original val ues after the project stops.

Required parameters:

- index: The security wall index to be set. The corresponding security wall needs to be adde d in the control software first. Value range: [1, 8].
- value:
  - true: Enable firewall.
  - false: Disable the firewall.

Example :

```
-- Enable the security wall with index 1 .
SetSafeWallEnable ( 1 , true )
```

Set WorkZoneEnable

prototype :

```
SetWorkZoneEnable ( index ,value)
```

describe :

Enables or disables the specified security zone. The security zone settings configured by

this command only take effect while the current project is running; they revert to their orig inal values after the project stops.

Required parameters:

- index: The index of the security zone to be set. The corresponding security zone needs to be added in the control software first. Value range: [1 , 6].
- v alue:
  - true: Enables the safe zone.
  - false: Close the safe zone.

Example :

```
-- Enable the safe zone with index 1 .
SetWorkZoneEnable ( 1 , true )
```

Se tCollisionLevel

prototype :

```
SetCollisionLevel ( level)
```

describe :

Sets the collision detection level. For CRA/CRAF models, this command sets the TCP forc e and power values for safety limits . The values set by this command only take effect du ring the current project run and revert to their original values after the project stops.

Required parameters:

Level : Collision detection level. 0 means collision detection is off, and the higher the number (1-5), the higher the sensitivity.

Example :

```
-- Set the collision detection level to 3 .
Se tCollisionLevel( 3 )
```

Se tBackDistance

prototype :

```
S etBackDistan ce(distance)
```

describe :

sets the distance the robotic arm should retract after detecting a collision. The value set o nly applies during the current project run; it reverts to its original value after the project sto ps.

Required parameters:

d istan ce: Collision back-off distance, range: [0,50], unit: mm.

Example :

```
-- Set the collision backoff distance to 20mm .
Se tBackDistance( 20 )
```

IO

Instruction List

I/ O commands are used to read and write I/O operations and set related parameters for the r obotic arm system.

instruction	Function
D I	Get the status of the DI port
DI Group	Get the status of multiple DI ports
D O	Set the status of the digital output port
D OGroup	Configure the status of multiple digital output ports
G etDO	Get the current status of the digital output port
G etDOGroup	Get the current status of multiple digital output ports
AI	Get the value of the AI port
A O	Set the value of the analog output port.
G etAO	Get the current value of the analog output port

D I

prototype :

```
D I ( index)
```

describe :

Read the status of the digital input port.

Required parameters:

in dex: The number of the DI terminal.

return :

The corresponding DI terminal status (ON/OFF).

Example :

```
-- When DI1 is ON , the robotic arm moves to point P1 in a linear motion .
if (DI( 1 )==ON) th
en
        Mo vL(P1)
en d
```

D IGroup

prototype :

```
D IGroup (index1, . .. , in dexN)
```

describe :

Read the status of multiple digital input ports.

Required parameters:

ind ex: The DI terminal number. Multiple numbers can be entered, separated by commas.

return :

the corresponding DI terminal is returned as an array.

Example :

```
-- When both DI1 and DI2 are ON , the robotic arm moves to point P1 in a linear motion .
local digroup = DIGroup( 1 , 2 )   if
(digroup[ 1 ]&digroup[ 2 ]==ON) then
        Mo vL(P1)
end
```

DO

prototype :

```
D O(index, ON |OFF, time) ms)
```

describe :

Configure the digital output port status.

Required parameters:

- index: The number of the DO terminal.
- ON|OFF: The DO port status to be set.

Optional parameters:

tim e_ms: Continuous output time, in milliseconds, range: [25, [60000]. If this parameter is set, the system will automatically invert the DO after the specified time . Inversion is an asynchronous action and will not block the instruction queue; the system will execute the next instruction immediately after executing the DO output. One instruction.

Example :

```
-- Set DO1 to ON .
DO ( 1 ,ON)
```

```
-- Set DO1 to ON, and then automatically set it to OFF after 50ms .
DO ( 1 ,ON, 50 )
```

D OGroup

prototype :

```
D OGroup({index1,ON|OFF},. .,{indexN,ON|OFF})
```

describe :

Configure the status of multiple digital output ports.

Required parameters:

- index: The number of the DO terminal.
- ON|OFF: The DO port status to be set.

can be set , each group is enclosed in curly braces and separated by commas.

Example :

```
-- Set DO1 and DO2 to ON .
DO Group({ 1 , ON}, { 2 , ON})
```

G etDO

prototype :

```
GetDO ( index)
```

describe :

Get the current status of the digital output port.

Required parameters:

ind ex: The number of the DO terminal.

return

The corresponding DI terminal status (ON/OFF).

Example :

```
-- Get the current state of DO1 .
local do1 = GetDO( 1 )
```

G etDOGroup

prototype :

```
G etDOGrou p(index1, . .. , in dexN)
```

describe :

Get the current status of multiple digital output ports.

Required parameters:

inde x: The number of the DO terminal. Multiple numbers can be entered, separated by com mas.

return :

the corresponding DO terminal is returned as an array.

Example :

```
-- Get the current state of DO1 and DO2 .
local dogroup = GetDOGroup( 1 , 2 ) local
do1 = dogroup[ 1 ]
local do2 = dogroup[ 2 ]
```

AI

prototype :

```
A I( index)
```

describe :

Read the value from the analog input port. The meaning of the value (voltage/current) ca n be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/AO** page of the c ontrol cabinet.

Required parameters:

i nd ex: The number of the AI terminal.

return :

The corresponding AI terminal value.

Example :

```
-- Read the value of AI1
local ai1 = AI( 1 )
```

AO

prototype :

```
A O(index x,value)
```

describe :

Configure the values for the analog output port. The meanings of these values (voltage/c urrent) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/AO** pag e of the control cabinet.

Required parameters:

- in dex: The number of the AO terminal.

- value: The value to be set. Voltage range: [0,10], unit: V; Current range: [4,20], unit: mA .

Example :

```
-- Set the output value of AO1 to 2 .
AO ( 1 , 2 )
```

G etAO

prototype :

```
Ge etAO (index)
```

describe :

Retrieves the current value at the analog output port. The meaning of the value (voltage/current) can be found in CBSH Studio . Pro's **monitoring >** View and modify the **AI/ AO pages of the control cabinet .**

Required parameters:

ind ex: The number of the AO terminal.

return

The corresponding AO terminal value.

Example :

```
-- Get the current state of AO1 .
local ao1 = GetAO( 1 )
```

End tools

Instruction List

End- effector commands are used to read and write the end-effector I/O of the robotic arm system and set related parameters.

instruction	Function
ToolDI	Read the status of the end digital input port
To olDO	Set the status of the terminal digital output port (queue com mand)
G e tToolDO	Get the current status of the terminal digital output port
T oolAI	Read the value of the terminal analog input port
Se tToolMode	Configure the communication mode of the terminal multiplex er
Ge tToolMode	Obtain the communication mode of the end multiplex termina l
Se tToolPower	Set the power supply status of the end tool
Se tTool485	Configure the data format corresponding to the RS485 interf ace of the terminal tool.

ToolDI

prototype :

```
TOOLD I ( index)
```

describe :

Read the status of the end digital input port.

Required parameters:

i nd ex: The number of the terminal DI.

return :

The corresponding DI terminal status (ON/OFF).

Example :

```
-- When the end effector DI1 is ON , the robotic arm moves to point P1 in a linear motion .
if (ToolDI( 1 )==ON) th
en
    Mo vL(P1)
en d
```

ToolDO

prototype :

```
T oolDO(index , ON|OFF)
```

describe :

Configure the status of the terminal digital output port.

Required parameters:

- index: The number of the DO terminal at the end.
- ON|OFF: The DO port status to be set.

Example :

```
-- Set the terminal DO1 to ON .
To olDO( 1 ,ON)
```

GetToolDO

prototype :

```
GetTool DO ( index)
```

describe :

Get the current status of the terminal digital output port.

Required parameters:

ind ex: The number of the DO terminal at the end.

return

The corresponding end DO terminal status (ON/OFF).

Example :

```
-- Get the current state of the terminal DO1 .
Ge tToolDO( 1 )
```

T oolAI

prototype :

```
ToolA I ( index)
```

describe :

Read the value of the terminal analog input port.

use , the terminal needs to be set to analog input mode via SetToolMode.

> illustrate :
> without an end-effector AI interface cannot use this interface.

Required parameters:

i nd ex: The number of the end AI terminal.

return :

The corresponding AI terminal value.

Example :

```
-- Read the value of the end AI1 and assign it to the variable test .
test = ToolAI( 1 )
```

SetToolMode

prototype :

```
SetToolMode (mode, type , identify)
```

describe :

For models where the AI1 and AI2 interfaces of the robotic arm end effector are multipl exed with the 485 interface, the end effector multiplexing terminal mode can be set thro ugh this interface .

The default mode is 485 mode.

> illustrate :
> that do not support end-effector mode switching.

Required parameters:

- mode: The mode of the multiplexed terminal.
  - 1:485 mode.
  - 2: Analog input mode.
- type:
  - This parameter is invalid when mode is 1.
  - When mode is 2, this parameter is used to set the mode of the analog input.

  The units digit represents the AI1 mode, the tens digit represents the AI2 mode, and when the tens digit is 0, only the units digit needs to be entered.

  Values :

  - 0: 0~10V voltage input mode.
  - 1: Current acquisition mode.
  - 2: 0~5V voltage input mode.

  Example :

  - 0: Both AI1 and AI2 are 0~10V voltage input modes.
  - 1: AI2 is a 0~10V voltage input mode, while AI1 is a current acquisition mode.
  - 11: Both AI2 and AI1 are current acquisition modes.
  - 12: AI2 is the current acquisition mode, and AI1 is the 0~5V voltage input mode.
  - 20: AI2 is a 0~5V voltage input mode, and AI1 is a 0~10V voltage input mode.

Optional parameters:

`iden tify`: Used to specify the end effector when the robotic arm has multiple end effectors. The default value is 1.

- 1 indicates the aircraft plug 1.
- 2 indicates the 2nd pendulum.

Example :

```
-- Set the end-to-end multiplexer terminal to 485 mode.
SetToolMode ( 1 , 0 )
```

```
-- Set the multiplexed terminal of the CR20A terminal connector 2 to 485 mode.
SetToolMode ( 1 , 0 , 2 )
```

```
-- Set the end multiplexer terminal to analog input mode, with both channels in 0~10V voltage input mode.
SetToolMode ( 2 , 0 )
```

```
-- Set the end multiplexer terminal to analog input mode. AI1 is 0~10V voltage input mode, and AI2 is current mode.
SetToolMode ( 2 , 10 )
```

G etToolMode

prototype :

```
GetToolMode ( identify)
```

describe :

Get the current mode of the end multiplexing terminal.

Optional parameters:

`iden tify`: Used to specify the end effector when the robotic arm has multiple end effectors. The default value is 1.

- 1 indicates the aircraft plug 1.
- 2 indicates the 2nd pendulum.

return :

- mode: The mode of the multiplexed terminal.
- type: The mode of simulated input. See the parameter with the same name in SetToolMode for details.

Example :

```
-- The mode for obtaining the end multiplexed terminal.
local mode, type = GetToolMode()
```

```
-- Obtain the mode of the CR20A end connector 2 multiplex terminal.
local mode, type = GetToolMode( 2 )
```

SetToolPower

prototype :

```
SetToolPower ( status)
```

describe :

This setting configures the power supply status of the end effector, typically used to restart the end effector power supply, such as re-powering and initializing the end effector grippers. If this interface is to be called continuously, it is recommended to wait at least 4ms between calls.

> **illustrate :**
> The terminal DO will also fail after the terminal power is turned off .

Required parameters:

sta tus: Power supply status of the end tool.

- 0: Power off.
- 1: Turn on the power.

Example :

```
-- Restart the power supply of the terminal tool.
SetToolPower ( 0 )
Wait ( 5 )
SetToolPower ( 1 )
```

SetTool485

prototype :

```
S etTool485(baud,parity, stopbit,identify)
```

describe :

Configure the data format corresponding to the RS485 interface of the terminal tool.

> illustrate :
> without an end effector 485 interface cannot use this interface.

Required parameters:

baud rate of RS485 interface.

Optional parameters:

- parity: Whether there is a parity bit.
  - "O": Odd parity.
  - "E": Even parity.
  - "N": No parity bit.
  - The default value is "N".
- stopbit: Stop bit length. Value range: 0.5, 1, 1.5, 2. Default value is 1.
  An error within ±0.1 will automatically select the nearest value (0.4001 takes a value of 0.5). 0.3999 Error message: 1.09999 Value 1).
- identify: When the robotic arm has multiple end effector attachments, it is used to specify the attachment, where 1 represents attachment 1 and 2 represents attachment 2.

Example :

```
-- Set the baud rate of the RS485 interface of the end tool to 115200Hz , with no parity bit and a stop bit
length of 1. ` SetTool485 ( 115200 , "N" , 1 )`
```

TCP & UDP

Instruction List

The TCP & UDP functions are used for TCP or UDP communication.

instruction	Function
TC PCreate	Create a TCP network object
TC PStart	Establish TCP connection
T CPRead	Receive data sent by the TCP peer
TC PWrite	Send data to the TCP peer
TC PDestroy	Disconnect the TCP connection and destroy the socket obj ect.
U D PCreate	Create a UDP network object
U DPRead	Receive data sent by the UDP peer
U D PWrite	Send data to the UDP peer

T C PCreate

prototype :

```
TCPCreate(isServer,  IP, p ort)
```

describe :

Creates a TCP network object; only one object can be created.

Required parameters:

- isServer: Whether to create a server.
  - true: indicates that a server is created.
  - false: indicates that a client is created.
- IP: Server IP address. It must be on the same network segment as the client IP address a nd must not conflict with it.
  - The server is created using the robotic arm's IP address.
  - The client is created using the address of the peer.
- port: Server port.

Do not use the ports listed below that are already in use by the system, as this will cause the server creation to fail.
7, 13, twenty two, 37, 139, 445, 502, 503 (Ports 0-1024 are custom ports defined by the Linux system and are likely to be occupied ; please avoid using them if possible).

1501, 1502, 1503, 4840, 8172, 9527,

11740, 22000, 22001, 29999, 30004, 30005,

30006, 60000~65504, 65506, 65511~65515,

65521, 65522.

return :

- err: 0 indicates that the TCP network object was created successfully. 1 indicates that the creation of the TCP network object failed.
- socket: The socket object that is created.

Example 1:

```
-- Create a TCP server.
local IP address: "192.168.5.1" -- The robotic arm's IP address is used as the
server's local IP address . port= 6001  -- server port
local  err= 0
local socket= 0
err, socket = TCPCreate( true , ip, port)
```

Example 2:

```
-- Create a TCP client.
local  IP address: "192.168.5.25"  -- The IP address of external devices, such as cameras,
is used as the local IP address of the server . port= 6001  -- server port
local  err= 0
local socket= 0
err, socket = TCPCreate( false , ip, port)
```

TCPStart

prototype :

```
T CPStar t(socket, t i meout)
```

describe :

Establish a TCP connection.

- When the robotic arm acts as a server, it waits for the client to connect.
- When the robotic arm acts as a client, it actively connects to the server.

Required parameters:

- socket: A socket object that has already been created.
- timeout: The timeout period, in seconds.
  - If set to 0, it will wait until the connection is successfully established.
  - If the value is not 0, the connection will fail after the set time has elapsed.

return :

Connection result.

- 0: Connection successful.
- 1: Input parameters are incorrect.
- 2: The socket object does not exist.
- 3: The timeout setting is incorrect.
- 4: Connection failed.

Example :

```
-- Initiate TCP connection establishment and wait until the connection is successfully established.
err = TCPStart(socket, 0 ) - - so cket is the socket object returned successfully by TCPCreate.
```

T CPRead

prototype :

```
T CPRead (socket, t im eout, t ype )
```

describe :

Receive data sent by the TCP peer.

Required parameters:

sock et: The socket object that has been created.

Optional parameters:

- timeout: The timeout period, in seconds.
  - If not set or set to a value less than or equal to 0, it will wait until all data has been re ad before proceeding.
  - If set to a value greater than 0, execution will continue directly after the set time has e lapsed.
- type: Return value type.
  - If not set or set to "table", the RecBuf cache format is in table form.
  - If set to "string", RecBuf will cache as a string.

return :

- err: 0 indicates successful data reception 1 indicates that data reception failed.
- Recbuf: Receive data buffer.

Example :

```
-- It receives TCP data without ever timing out, and the received
data is saved in table format.
-- The socket is the socket object returned successfully by TCPCreate.
err, RecBuf = TCPRead(socket) - - The data type of Re cBuf is table.
-- Receive TCP data with a timeout of 5 seconds, and save the
received data in table format.
-- The socket is the socket object returned successfully by TCPCreate.
err, RecBuf = TCPRead(socket, 5 ) -- The RecBuf data type is table.
```

```
-- Receives TCP data without timeout, and saves the
received data as a string .
-- The socket is the socket object returned successfully by
TCPCreate.
err, RecBuf = TCPRead(socket, 0 , "string" ) - - The data type of Re cBuf is a string.
```

TCPWrite

prototype :

```
T CPWrit e(socket, b uf, t im eout)
```

describe :

Send data to the TCP peer.

Required parameters:

- socket: A socket object that has already been created.
- buf: The data to be sent.

Optional parameters:

timeout : The timeout period in seconds .

- If not set or set to 0, it will wait until the other end has received all the data before procee ding.
- If it is not 0, it will continue to execute directly after the set time has elapsed.

return :

Send the result.

- 0: Sent successfully.
- 1: Sending failed.

Example :

```
-- Send TCP data containing "test", with no timeout.
TCPWrite(socket, "test " -- `s ocket` refers to the socket object returned successfully by TCPCreate.
```

```
-- Send TCP data with the content "test" and a timeout of 5 seconds.
TCPWrite(socket, "test" , 5 ) -- `s ocket` refers to the socket object returned successfully by TCPCreate.
```

TCPDestroy

prototype :

```
T CPDestro y(socket)
```

describe :

Disconnect the TCP connection and destroy the socket object.

Required parameters:

sock et: The socket object that has been created.

return :

Execution result.

- 0: Execution successful.
- 1: Execution failed.

Example :

```
-- Disconnect from the TCP peer.
TCPDestroy(socket) - - so cket is the socket object returned successfully by TCPCreate.
```

UDPCreate

prototype :

```
UDPCreate(isServer, IP, p ort)
```

describe :

Creates a UDP network object; only one object can be created.

Required parameters:

- isServer: Whether to create a server.
  - true: indicates that a server is created.
  - false: indicates that a client is created.
- IP: Enter the peer's IP address when creating both the server and client. It must be on the same network segment as the robotic arm's IP address and must not conflict with it.
- port:
  - When creating a server , it indicates that both the local and remote ends will use this port. Do not use ports that are already in use by the system; see the T CPCreate parameter description for details.
  - When creating a client, specify the port of the peer. The local end will then use a random port when sending data.

return :

- err: 0 indicates that the UDP network object was created successfully. 1 indicates that the creation of the UDP network object failed.
- socket: The socket object that is created.

Example 1:

```
-- Create a UDP server.
local IP address: "192.168.5.25" -- The IP address of an external device, such as a camera, is used as the local IP address of the peer device . port= 6001 -- Both the local and remote ends use this port.
local err= 0
local socket= 0
err, socket = UDPCreate( true , ip, port)
```

Example 2:

```
-- Create a UDP client.
local IP address: "192.168.5.25" -- The IP address of an external device, such as a camera, is used as the local IP address of the peer device . port= 6001 -- peer port
local err= 0
local socket= 0
err, socket = UDPCreate( false , ip, port)
```

UDPRead

prototype :

```
U DPRead (socket, t im eout, t ype )
```

describe :

Receive data sent by the UDP peer.

Required parameters:

sock et: The socket object that has been created.

Optional parameters:

- timeout: The timeout period, in seconds.
  - If not set or set to a value less than or equal to 0, it will wait until all data has been re ad before proceeding.
  - If set to a value greater than 0, execution will continue directly after the set time has e lapsed.
- type: Return value type.
  - If not set or set to "table", the RecBuf cache format is in table form.
  - If set to "string", RecBuf will cache as a string.

return :

- err: 0 indicates successful data reception 1 indicates that data reception failed.
- Recbuf: Receive data buffer.

Example :

```
-- Receive UDP data, and save the received data as both a string
and a table .
-- The socket is the socket object returned successfully by UDPCreate.
err, RecBuf = UDPRead(socket, 0 , "string" ) - - The RecBuf data type is the string 'err',
RecBuf = UDPRead(socket, 0 ) -- The RecBuf data type is table.
```

UDPWrite

prototype :

```
U DPWrit e(socket, b uf, t im eout)
```

describe :

Send data to the UDP peer.

Required parameters:

- socket: A socket object that has already been created.
- buf: The data to be sent.

Optional parameters:

timeout : The timeout period in seconds .

- If not set or set to 0, it will wait until the other end has received all the data before procee ding.
- If it is not 0, it will continue to execute directly after the set time has elapsed.

return :

Send the result.

- 0: Sent successfully.
- 1: Sending failed.

Example :

```
-- Send UDP data with the content "test" .
UDPWrite(socket, "test " -- s ocket is the socket object returned successfully by UDPCreate.
```

Modbus

Instruction List

Modbus function is used to establish communication between the Modbus master and sl ave. For the range and definition of the register address, please refer to the Modbus regi ster address definition description of the corresponding slave .

instruction	Function
ModbusCreate	Create a Modbus master station
Mo dbusRTUCreate	Create a Modbus master station based on an RS485 interface
ModbusClose	Disconnect from Modbus slave station
G etInBits	Read contact register
Ge tInRegs	Read input register
GetCoils	Read coil register
Se tCoils	Write to coil register
Ge tHoldRegs	Read holding register
Se tHoldRegs	Write to save register

various registers follow the standard Modbus protocol:

Register type	Read registers	Write a single register	Write multiple registers
Coil Register	0 1	0 5	0 F
Contact register	0 2	-	-
Input register	0 4	-	-
Holding register	0 3	0 6	1 0

ModbusCreate

prototype :

```
Modbus Create (IP, port, slave_id, isRTU)
```

describe :

Create a Modbus master station based on the TCP/IP protocol and establish connections with slave stations. Up to 15 devices can be connected simultaneously.
connecting to the robot's built-in slave station, set the IP address to the robot's IP address (default 192.168.5.1, which can be modified), and set the port to 502 (map1) or 1502 (map2). See Appendix A for details. Modbus register definition .

connecting to a third-party slave station, the IP address and port are the addresses of the third-party slave station. For the range and definition of register address values when reading and writing registers, please refer to the Modbus register address definition description of the corresponding slave station.

Required parameters:

- IP: The IP address of the slave station.
- port: slave port.

Optional parameters:

- slave_id : Subsite ID.
- isRTU: Boolean value.
  - When isRTU is false, a Modbus connection is established based on the control cabinet network port. TCP communication.
  - When isRTU is true, a Modbus based on the RS485 terminal of the body is established. RTU communication can only use port 60000.

> ⚠ **Notice :**
> This parameter determines the protocol format used for data transmission after the connection is established, but it does not affect the connection result. Therefore, if this parameter is set incorrectly when creating the master station, the station can still be created successfully, but subsequent communication will result in errors.

return :

- err:
  - 0: Modbus master station created successfully .
  - 1: The maximum number of main sites created has been reached. Failed to create a new main site.
  - 2: The main site failed to initialize. It is recommended to check the IP, port, and network conditions.
  - 3: Connection to slave station failed. It is recommended to check whether the slave station was established normally and whether the network is normal.
- id: The returned master station index, used when calling other Modbus commands subsequently. Value range: [0, ... 14).

Example :

```
-- Create a ModbusTCP master station and establish a
connection with the robot's built-in slave station.
--   The IP address is the robot's IP address , the port is 502 , and
the slave ID is not specified .          local IP address:
"192.168.5.1"
local port=502
local  err=0     local
id=0
err, id = ModbusCreate(ip, port)
```

```
-- Create a Modbu sTCP master and establish a
connection with the specified slave.
-- The slave IP is 192.168.5.123 , the port is 503 , and the
slave ID is 1 (local). IP address: "192.168.5.123"
local port= 503
local  err= 0
local id= 0
err, id = ModbusCreate(ip, port, 1)
```

```
-- Create a Mod busRTU-over-TCP master and establish a
connection with the specified slave.
-- The slave IP is 192.168.5.123 , the port is 503 , and the slave ID
is 1 .       local IP address: "192.168.5.123"
local port= 503
local  err= 0
local id= 0
err, id = ModbusCreate(ip, port, 1 , true )
```

Mo dbusRTUCreate

prototype :

```
ModbusRTUCr eate(slave_id, baud, p a rity, d ata_bit, stop_bit)
```

describe :

Create a Modbus master station based on the control cabinet's RS485 interface and establish connections with slave stations. Up to 15 devices can be connected simultaneously.

Required parameters:

- slave_id : Subsite ID.
- baud: The baud rate of the RS485 interface.

Optional parameters:

- parity: Whether there is a parity bit. "O" indicates odd parity, "E" indicates even parity, and "N" indicates no parity bit. The default value is "E" when no parameter is specified.
- da ta_` bit`: Length of data bits. Value range: 8. The default value is 8 when no parameter is specified.
- `s topbit`: Stop bit length. Value range: 1, 2. Default value is 1 when no parameter is specified.

return :

- err: 0 indicates that the Modbus master station was created successfully. 1 indicates that the creation of the Modbus master station failed.
- id: The returned master station index, which will be used when calling other Modbus commands later.

Example :

```
-- Create a Modbus master station and establish a connection with the slave station connected to the RS485 interface. The slave station ID is 1 and the baud rate is 115200 .
err, id = ModbusRTUCreate( 1 , 115200 )
```

```
-- Create a Modbus master station and establish a connection with the slave station connected to the RS485 interface. The slave station ID is 1 , the baud rate is 115200 , there is no parity bit, and the data bit length is... The value is 1 , and the stop bit length is 2 .
err, id = ModbusRTUCreate( 1 , 115200 "N" , 8 , 2 )
```

ModbusClose

prototype :

```
ModbusC lose( id )
```

describe :

Disconnect from the Modbus slave and release the master station .

Required parameters:

i d: The main site index that has been created.

return :

Operation results.

- 0: Connection successfully disconnected.
- 1: Connection failed.

Example :

```
-- Disconnect from the Modbus slave.
ModbusClose (id)
```

G etInBits

prototype :

```
Ge etIn Bits(id, addr,  count)
```

describe :

Read the value of the Modbus slave contact register address. This corresponds to Modbus fu nction code 02.

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the contact register.
- count: The number of consecutive reads of the contact register. Value range: [1, ... [2000] (Modbus TCP protocol limitation; the actual value range should be determined based on the number of slave registers or protocol specifications).

return :

the contact register address is stored in a table. The first value in the table corresponds to the value of the starting address of the contact register.

Example :

```
-- Read the values of 5 consecutive addresses starting from address 0 .
inBits = GetInBits(id, 0 , 5 )
```

G etInRegs

prototype :

```
Ge etIn Regs(id, addr,  count, type )
```

describe :

according to the specified data type. Corresponds to Modbus function code 04.

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the input register.
- count: The number of consecutive reads from the input register. Value range: [1, ... 125] (Modbus TCP protocol limitations; the actual value range should be determine d based on the number of slave registers or protocol specifications).

Optional parameters: type: Data type.

- If empty, the default value is U16.
- U16: 16-bit unsigned integer (2 bytes, occupying 1 register).
- U32: 32-bit unsigned integer (4 bytes, occupying 2 registers).
- F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers).
- F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers).

return :

the input register address is stored in a table. The first value in the table corresponds to the va lue of the starting address of the input register.

Example :

```
-- Read a 16 -bit unsigned integer starting at address 2048 .
data = GetInRegs(id, 2048    1 )
```

```
-- Read a 32 -bit unsigned integer starting at address
2048. `data = GetInRegs(id, 2048 , 2 , ...) ` "U32 "
```

```
-- Read two 32- bit single-precision floating-point
numbers starting at address 2048. `data = GetInRegs(id,
2048 , 4 , ...) ` "F32 "
```

G etCoils

prototype :

```
GetCo ils(id, a ddr , c ount)
```

describe :

Read the value of the Modbus slave coil register address. Corresponds to Modbus function c ode 01.

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the coil register.
- count: The number of consecutive reads from the coil register. Value range: [1, ... [2 000] (Modbus TCP protocol limitation; the actual value range should be determined based on the number of slave registers or protocol specifications).

return :

the coil register address is stored in a table. The first value in the table corresponds to the star ting address of the coil register.

Example :

```
-- Read the values of 5 consecutive addresses starting from address 0 .
Coils = GetCoils(id, 0 , 5 )
```

SetCoils

prototype :

```
S etCo ils(id, a ddr, c ount, t able )
```

describe :

Writes the specified value to the specified address in the coil register. This corresponds to M odbus function codes 05 (write single) and 0F (write multiple).

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the coil register.

- count: The number of consecutive writes to the coil register. Value range: [1, ... [1968] (Modbus TCP protocol limitation; the actual value range should be determined based on the number of slave registers or protocol specifications).
- table: The values to be written to the coil register are stored in the table. The first value in the table corresponds to the starting address of the coil register .

return :

The return value is 0 or -1, where 0 indicates that the setting was successful and -1 indicates that the setting failed.

Example :

```
-- Start writing five coils consecutively from address 1024 .
local coils = { 0 , 1 , 1 , 1 , 0 }
SetCoils(id, 1024 , #coils, coils)
```

GetHoldRegs

prototype :

```
G etHold Regs(id, a ddr, count , type )
```

describe :

according to the specified data type. Corresponds to Modbus function code 03.

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the holding register.
- count: The number of consecutive reads from the holding register. Value range: [1, ... 125] (Modbus TCP protocol limitations; the actual value range should be determined based on the number of slave registers or protocol specifications).

Optional parameters:

ty pe: Data type.

- If empty, the default value is U16.
- U16: 16-bit unsigned integer (2 bytes, occupying 1 register).
- U32: 32-bit unsigned integer (4 bytes, occupying 2 registers).
- F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers).
- F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers).

return :

the hold register address is stored in a table. The first value in the table corresponds to the val ue at the starting address of the hold register.

Example :

```
-- Read a 16 -bit unsigned integer starting at address 2048 .
data = GetHoldRegs(id, 2048  1 )
```

```
-- Read a 32 -bit unsigned integer starting at address
2048. `data = GetHoldRegs(id, 2048 , 2 , ...) ` "U32 "
```

```
-- Read two 32- bit single-precision floating-point
numbers starting at address 2048. ` data =
GetHoldRegs(id, 2048 , 4 , ...) ` "F32 "
```

SetHoldRegs

prototype :

```
S etHold Regs(id, a ddr, count , t able , type )
```

describe :

according to the specified data type. This corresponds to Modbus function codes 06 (write a single value) and 10 (write multiple values).

Required parameters:

- id : The main site index that has been created.
- a ddr: The starting address of the holding register.
- count: The number of consecutive writes to the holding register. Value range: [1, ... [123] (Modbus TCP protocol limitations; the actual value range should be determin ed based on the number of slave registers or protocol specifications).
- table: The values to be written to the holding register are stored in the table. The fir st value in the table corresponds to the value at the starting address of the holding register .

Optional parameters:

ty pe: Data type.

- If empty, the default value is U16.
- U16: 16-bit unsigned integer (2 bytes, occupying 1 register).
- U32: 32-bit unsigned integer (4 bytes, occupying 2 registers).
- F32: 32-bit single-precision floating-point number (4 bytes, occupying 2 registers).
- F64: 64-bit double-precision floating-point number (8 bytes, occupying 4 registers).

return :

The return value is 0 or -1, where 0 indicates that the setting was successful and -1 indicate s that the setting failed.

Example :

```
-- Write a 16 -bit unsigned integer starting at address 2048 .
local data = { 12 }
SetHoldRegs(id, 2048 , 1 , data)
```

```
-- Write a 64 -bit double-precision floating-point number starting at address 2048 .
local data = { 95.32105 }
SetHoldRegs(id, 2048 , 4 , data, "F64 "
```

```
-- Starting at address 2048 , write two 64- bit double-precision floating-point numbers.
local data = { 95.32105 , 104.24411
SetHoldRegs(id, 2048 , 8 , data, "F64 "
```

Bus Register

Instruction List

Bus register instructions are used to read and write Profinet or Ethernet/IP bus registers.

> **illustrate :**
> Magician E6 does not support this set of instructions.

instruction	Function
Ge tInputBool	Get the boolean value at the specified address of the in put register.
G etInputInt	Get the int value at the specified address of the input r egister.
Ge tInputFloat	Get the float value at the specified address of the input register.
Ge tOutputBool	Get the boolean value at the specified address of the o utput register

Ge tOutputInt	Get the int value at the specified address of the output register
Ge tOutputFloat	Get the float value at the specified address of the output register
Se tOutputBool	Set the boolean value at the specified address of the output register.
Se tOutputInt	Sets the int value at the specified address of the output register.
Set OutputFloat	Set the float value at the specified address of the output register.

G etInputBool

prototype :

```
G etInputBoo l(address)
```

describe :

Retrieves the boolean value at the specified address of the input register.

Required parameters:

addr ess: Register address, value range [0-63].

return :

specified register address is either 0 or 1.

Example :

```
-- When the value of input register 0 is 1 , the subsequent operation is executed.
if (GetInputBool( 0 )== 1 ) th
en
        -- Perform subsequent operations
en d
```

G etInputInt

prototype :

```
G etInputIn t(address)
```

describe :

Retrieves the integer value at the specified address in the input register.

Required parameters:

addr ess: Register address, value range [0-23].

return :

the specified register address is an integer (int32).

Example :

```
-- Read the value of input register 1 and assign it to the variable regInt .
local regInt = GetInputInt( 1 )
```

G etInputFloat

prototype :

```
G etInputFlo at(address)
```

describe :

Retrieves the float value at the specified address of the input register.

Required parameters:

addr ess: Register address, value range [0-23].

return :

the specified register address is a single-precision floating-point number (float).

Example :

```
-- Read the value of input register 2 and assign it to the variable regFloat .
local regFloat = GetInputFloat( 2 )
```

G e tOutputBool

prototype :

```
G etOutputBo ol(address)
```

describe :

Retrieves the boolean value at the specified address of the output register.

Required parameters:

addr ess: Register address, value range [0-63].

return :

specified register address is either 0 or 1.

Example :

```
-- When the value of output register 0 is 1 , the subsequent operation is executed.
if (GetOutputBool( 0 )== 1 ) th
en
    -- Perform subsequent operations
en d
```

GetOutputInt

prototype :

```
G etOutputIn t(address)
```

describe :

Retrieves the integer value at the specified address of the output register.

Required parameters:

addr ess: Register address, value range [0-23].

return :

the specified register address is an integer (int32).

Example :

```
-- Read the value of output register 1 and assign it to the variable regInt .
local regInt = GetOutputInt( 1 )
```

GetOutputFloat

prototype :

```
G etOutputFlo at(address)
```

describe :

Retrieves the value of type float at the specified address of the output register.

Required parameters:

addr ess: Register address, value range [0-23].

return :

the specified register address is a single-precision floating-point number (float).

Example :

```
-- Read the value of output register 2 and assign it to the variable regFloat .
local regFloat = GetOutputFloat( 2 )
```

S etOutputBool

prototype :

```
SetOutputBool(address,   value)
```

describe :

Sets the output register to a boolean value at the specified address.

Required parameters:

- address: Register address, with a value range of [0-63].
- value: The value to set, which can be a boolean or 0/1.

Example :

```
-- Set the value of output register 0 to false.
Set tOutputBool( 0 , 0 )
```

SetOutputInt

prototype :

```
S etOutputIn t(address, value )
```

describe :

Sets the output register to a specified address containing an integer value.

Required parameters:

- address: Register address, with a value range of [0-23].
- value: The value to be set, which supports integers (int32).

Example :

```
-- Set the value of output register 1 to 123 .
Set tOutputInt( 1 , 123 )
```

SetOutputFloat

prototype :

```
S etOutputFlo at(address, v alue)
```

describe :

Sets the value of type float at the specified address of the output register.

Required parameters:

- address: Register address, with a value range of [0-23].
- value: The value to be set, supporting single-precision floating-point numbe rs (float).

  Due to the limitations of the floating-point storage mechanism (IEEE 7 54), single-precision floating-point numbers can only store approximat ely 6 to 7 significant digits (regardless of the decimal point position). Storing values with more than 6 significant digits as single-precision flo ating-point numbers may introduce inaccuracies; the more significant d igits , the greater the potential for inaccuracies.

470

Example :

```
-- Set the value of output register 2 to 12.3 .
Set tOutputFloat( 2 , 12.3 )
```