

Gorilla Studio Robot Software User Manual



CONTENTS

CONTENTS	2
Preface	9
1. Introduction	11
1.1. Overview of Gorilla Studio	11
1.2. Operating Environment	12
1.3. Connecting to the Network Port	12
1.4. Login	13
1.5. Interface	14
1.5.1. Introduction to the Main Interface	14
1.5.2. Navigation Bar	16
1.5.3. Toolbar	17
1.5.4. Editing Area	19
1.5.5. Shortcut Commands	20
1.5.6. Compile Information	21
1.5.7. Virtual Keyboard	22
1.5.8. Monitoring	23
1.6. Basic Terminology	30
1.6.1. Model	30
1.6.2. Coordinate System	31
1.7. Basic Robot Operations	35
1.7.1. User Permissions Description	35
1.7.2. Control Device Switching	38
1.7.3. Clear Fault	40
1.7.4. Coordinate System Switching	40
1.7.5. Tool Number Switching	42
1.7.6. Work Object Switching	43
1.7.7. Speed Adjustment	44
1.7.8. Jog	46
2. Settings	52
2.1. Supported Robot Models	52
2.1.1. SCARA	52
2.1.2. DELTA	52
2.1.3. 6-Axis Robot	53
2.1.4. DIAMOND	56
2.2. Robot Settings	57
2.2.1. Basic Information	57
2.2.2. Parameter Configuration	58
2.2.3. Axis Parameter Configuration	69

2.3. External Axis Settings	72
2.3.1. Axis Parameters for External Axes	72
2.3.2. Axis Constraints for External Axes	73
2.3.3. Axis Motion Constraints for External Axes	74
2.4. Calibration	75
2.4.1. Tool Coordinate System Calibration	75
2.4.2. Work Object Coordinate System Calibration	84
2.4.3. User Coordinate System Calibration	88
2.4.4. Kinematics Calibration	93
2.4.5. Robot Zero Position Calibration	95
2.4.6. Zero Offset	96
2.4.7. External Axis Zero Position Calibration	97
2.5. Multi-tasking	98
2.5.1. Overview	98
2.5.2. Task Activation	99
2.5.3. Task Start and Stop	100
2.5.4. Inter-Task Communication	100
2.5.5. Task Monitoring	101
2.6. Remote Control	101
2.6.1. Teach Pendant Control	102
2.6.2. IO Control	103
2.6.3. Modbus Control	105
2.6.4. PLC Control	107
2.7. Communication Settings	110
2.7.1. Network Configuration	110
2.7.2. SocketTCP Configuration	110
2.7.3. Modbus Configuration	113
2.7.4. Serial Port Settings	117
2.8. System Settings	118
2.8.1. System Information	118
2.8.2. General	118
2.8.3. Backup and Restore	124
2.8.4. Plugin Management	125
3. Variable Table	127
3.1. Global Variables	127
3.1.1. Overview	127
3.1.2. Edit Variable	133
3.2. Tool Coordinate System	139
3.2.1. Tool	139
3.2.2. Tool Center Point	140
3.2.3. Tool Coordinate System	141
3.2.4. Tool Load Parameters	142
3.2.5. External Tool	142
3.2.6. Using Tools	144

3.3. Work Object Coordinate System	147
3.3.1. Work Object	147
3.3.2. Work Object Coordinate System	150
3.3.3. Using Work Objects	151
3.4. User Coordinate System	153
3.5. Speed Setting Table	154
3.6. Acceleration Setting Table	156
3.7. Turn Area Dimension Setting Table	158
4. Editor	160
4.1. Program File List	160
4.1.1. Folder Operations	161
4.1.2. Program File Operations	162
4.2. Program File Editing	164
4.2.1. Example	164
4.2.2. Basic Operations	166
4.2.3. Local Variable Declaration	166
4.2.4. Program Editing	174
4.3. Save	177
4.4. Compile	177
4.5. Debugging	178
4.5.1. Program Status	179
4.5.2. Basic Operations	179
4.5.3. Loop Mode	181
4.5.4. Run Modes	181
4.5.5. Point Debugging	182
4.5.6. Task Type	182
4.6. Variable Monitoring	183
4.6.1. Local Variable Monitoring	183
4.6.2. Global Variable Monitoring	186
5. Logs	189
5.1. Filter Logs	189
5.2. Clear Logs	190
6. Help	191
6.1. Error ID Documentation	191
6.2. Manual	191
7. Command	193
7.1. Variables	193
7.1.1. Basic Types	193
7.1.2. Complex Types	194
7.2. Logical Command	200
7.2.1. If	200
7.2.2. For	201
7.2.3. While	202
7.2.4. Goto	203

7.2.5. Exit.....	203
7.2.6. Continue.....	204
7.2.7. Not.....	205
7.2.8. And.....	205
7.2.9. Xor.....	205
7.2.10. Or.....	205
7.2.11. Wait.....	206
7.2.12. Pause.....	206
7.2.13. Resume.....	206
7.2.14. Case.....	207
7.3. Motion Command.....	208
7.3.1. MoveL.....	208
7.3.2. MoveJ.....	212
7.3.3. MoveC.....	215
7.3.4. MoveAbsJ.....	218
7.3.5. MoveLJump.....	221
7.3.6. MovePath.....	224
7.3.7. AutoLJM.....	226
7.4. Mathematical Command.....	226
7.4.1. Sin.....	226
7.4.2. Cos.....	227
7.4.3. Tan.....	227
7.4.4. Asin.....	228
7.4.5. Acos.....	228
7.4.6. Atan.....	229
7.4.7. Exp.....	229
7.4.8. Ln.....	230
7.4.9. Log10.....	230
7.4.10. Pow.....	231
7.4.11. Sqrt.....	231
7.4.12. Ceil.....	232
7.4.13. Floor.....	232
7.4.14. Round.....	233
7.4.15. Min.....	234
7.4.16. Max.....	234
7.4.17. Abs.....	235
7.4.18. Sign.....	235
7.5. IO Command.....	236
7.5.1. SetDO.....	236
7.5.2. GetDI.....	236
7.5.3. PulseDO.....	237
7.5.4. Group.....	237
7.5.5. SetOG.....	238
7.5.6. GetIG.....	238

7.6. Program Command	239
7.6.1. Call	239
7.7. SocketTCP Command	239
7.7.1. SocketCreate	239
7.7.2. SocketClose	240
7.7.3. SocketSendString	240
7.7.4. SocketReadString	241
7.7.5. SocketStatus	241
7.8. String Command	242
7.8.1. StrConcat	242
7.8.2. StrDelete	242
7.8.3. StrFind	243
7.8.4. StrInsert	243
7.8.5. StrLeft	244
7.8.6. StrLen	244
7.8.7. StrMid	245
7.8.8. StrReplace	245
7.8.9. StrRight	246
7.8.10. StrToDouble	246
7.8.11. StrToInt	247
7.8.12. StrToWord	247
7.8.13. StrToFloat	248
7.8.14. DoubleToStr	248
7.8.15. DoubleToInt	249
7.8.16. DoubleToWord	249
7.8.17. DoubleToFloat	250
7.9. Pallet Command	250
7.9.1. SetPallet	250
7.9.2. SetPallet4	251
7.9.3. Pallet	253
7.9.4. PalletSequence	253
7.10. Modbus Command	254
7.10.1. GetModbusCoilToBool (Deprecated)	254
7.10.2. SetModbusCoilByBool (Deprecated)	255
7.10.3. GetModbusRegToWord (Deprecated)	255
7.10.4. SetModbusRegByWord (Deprecated)	256
7.10.5. GetModbusRegToInt (Deprecated)	257
7.10.6. SetModbusRegByInt (Deprecated)	257
7.10.7. GetModbusRegToDouble (Deprecated)	258
7.10.8. SetModbusRegByDouble (Deprecated)	259
7.10.9. GetModbusRegToFloat (Deprecated)	259
7.10.10. SetModbusRegByFloat (Deprecated)	260
7.10.11. ModbusTcpStatus (Deprecated)	260
7.10.12. ModbusStatus	261

7.10.13. GetModbusCoil	261
7.10.14. SetModbusCoil	262
7.10.15. GetModbusReg	263
7.10.16. SetModbusReg	263
7.11. Serial Port Command	264
7.11.1. ComOpen	264
7.11.2. ComClose	265
7.11.3. ComSendString	265
7.11.4. ComReadString	266
7.11.5. ComStatus	266
7.12. Tracking Command	267
7.12.1. ConveyorVision	267
7.12.2. ConveyorSensor	268
7.12.3. GetConveyorObject	269
7.12.4. TrackConveyor	269
7.12.5. TrackBase	270
7.13. Profinet Command	270
7.13.1. GetProfinetToWord	270
7.13.2. SetProfinetByWord	271
7.13.3. GetProfinetToInt	271
7.13.4. SetProfinetByInt	272
7.13.5. GetProfinetToDouble	272
7.13.6. SetProfinetByDouble	273
7.13.7. ProfinetStatus	273
7.14. Fly Camera Command	274
7.14.1. InitialFly	274
7.14.2. SetFly	274
7.14.3. StartFly	274
7.14.4. StopFly	275
7.14.5. CheckFly	275
7.14.6. GetFlyRobotPose	275
7.14.7. CalculationFlyResult	276
7.15. Other Commands	277
7.15.1. GetCartesianPose	277
7.15.2. GetAxisPosition	277
7.15.3. GetRobotTarget	278
7.15.4. GetRobotStatus	278
7.15.5. GetRunStatus	279
7.15.6. ZeroCalibration	280
7.15.7. UToEuler	280
7.15.8. EulerToU	281
7.15.9. BitSet	281
7.15.10. BitCheck	282
7.15.11. PoseMultiplication	283

7.15.12. PoseInverse	284
7.15.13. TimeStart	284
7.15.14. TimeOut	285
7.15.15. Print	286
8. Plugin Extension	287
8.1. Vision Calibration	287
8.1.1. Overview	287
8.1.2. Camera Installation	287
8.1.3. Calibration Steps	288
8.2. Conveyor	294
8.2.1. Overview	294
8.2.2. Hardware Configuration	297
8.2.3. Coordinate System Settings	299
8.2.4. Parameter Settings	303
8.2.5. Program Examples	314
8.2.6. Frequently Asked Questions (FAQ)	316
8.3. Fly Camera	316
8.3.1. Overview	316
8.3.2. Basic Parameters	317
8.3.3. Fly Camera Calibration	319
8.3.4. Program Example	320
9. Teaching Pendant	323
9.1. Teaching Pendant Appearance	323
9.1.1. Mode Switch	324
9.1.2. Emergency Stop Button	324
9.1.3. Safety Switch	324
9.1.4. Axis Operation Keys	324
9.2. Teaching Pendant Interface	325
9.2.1. Login Interface	325
9.2.2. Main Interface	327
9.2.3. Physical Hardware Buttons	327
10. Appendix	330
10.1. Modbus Slave Address Table	330
10.2. PLC Address Table	335
10.3. SCARA 4-Axis Torque Homing Instructions	336
10.3.1. Hardware Preparation	336
10.3.2. Installation	336
10.3.3. Homing	337

Preface

Disclaimer

The information contained in this manual is subject to change without prior notice and should not be regarded as a commitment by Hangzhou Robotphoenix Industrial Robotics Co., Ltd. (hereinafter referred to as Robotphoenix). Except as expressly stated anywhere in this manual, nothing in this manual shall be regarded as any type of guarantee or warranty made by Robotphoenix for loss of personnel or property of the user, personal injury or property damage, fitness for a particular purpose, or similar matters.

Copyright Statement

This manual and any part thereof may not be reproduced or copied, disclosed to a third party, or used for any unauthorized purpose without the written permission of Hangzhou Robotphoenix Industrial Robotics Co., Ltd. Violation of this clause will be subject to prosecution.

Document Overview

This manual introduces the basic operations, quick start guide, program editing, commissioning and running, parameter settings, communication service management, and other related topics of Gorilla Studio programming software.

Version Change Record

Revision Date	Release Version	Changes Made
2024-10	V1.10	
2024-11	V1.11	Add array function introduction Add For command introduction Add Case command introduction Add Group, SetOG, GetIG command introduction Add UToEuler, EulerToU, BitSet, BitCheck command introduction Add GetModbusRegToFloat, SetModbusRegByFloat, GetModbusCoil, SetModbusCoil, GetModbusReg,

		SetModbusReg command introduction Revise teaching introduction Add robot parameters: enable U-axis, disc orientation following Add axis parameters: homing torque limit, homing final position Revise backup/restore introduction Revise allowable range for axis and external axis limit coefficients Add axis parameters: torque limit, torque limit coefficient Add ZeroCalibration command introduction
2025-1	V1.12	Revise the Profinet command introduction Add the fly camera function introduction
2025-2	V1.13	Revise Modbus configuration-related introduction Add serial port configuration introduction Add PalletSequence command introduction Add Modbus command introduction Add serial port command introduction Add fly camera command introduction Add software update and license clearing introduction Revise vel/acc-related introduction Add point debugging and task type introduction
2025-3	V1.14	Replace Modbus slave address table Replace interface screenshots Add pose transformation command introduction Add command to get robot pose introduction Add conveyor program example Add fly camera program example
2025-6	V1.17	Add GetRobotTarget command and introduction Add TimeStart command and introduction Add TimeOut command and introduction Add Modbus slave data addresses Revise Modbus remote control synchronous display Add multi-plugin upload and uninstall function Add sequence number download/upload function Revise pallet quantity
2025-8	V1.18	Replace interface with V18 version Add motion speed shortcut key Add multi-task quantity Make global variable arrays editable, with variable names and same-level comments Add Print command introduction Add SetPallet4 command introduction Add actual axis torque limit parameter function Add SCARA Python model singularity smoothing function Add SCARA and DELTA model tool coordinate system

		calibration using multi-point method Add third-segment limit function for DELTA model Add conveyor tracking encoder resolution multi-point calibration and visual angle inversion function Add operation log recording function and detailed error reporting Add GetRunStatus command Add remote PLC control Add Python model kinematics calibration introduction Revise vel/acc-related introduction
--	--	--

1. Introduction

1.1. Overview of Gorilla Studio

Gorilla Studio robot control system is suitable for the full range of robots launched by Robotphoenix and provides functions such as programming, commissioning, robot parameter configuration, status monitoring, and robot motion control.

In programming, Gorilla Studio is case-sensitive and allows direct text input via the keyboard, supporting both physical and virtual keyboards. During programming, it supports user-defined variables, code completion, and secondary hints, which greatly improve programming efficiency.

In commissioning, the currently running line is highlighted. Multiple program files can be opened simultaneously, and switching between different files is supported. Multi-task operation is supported. During commissioning, local variables and global variables can be monitored, and variable values can be modified while the program is running.

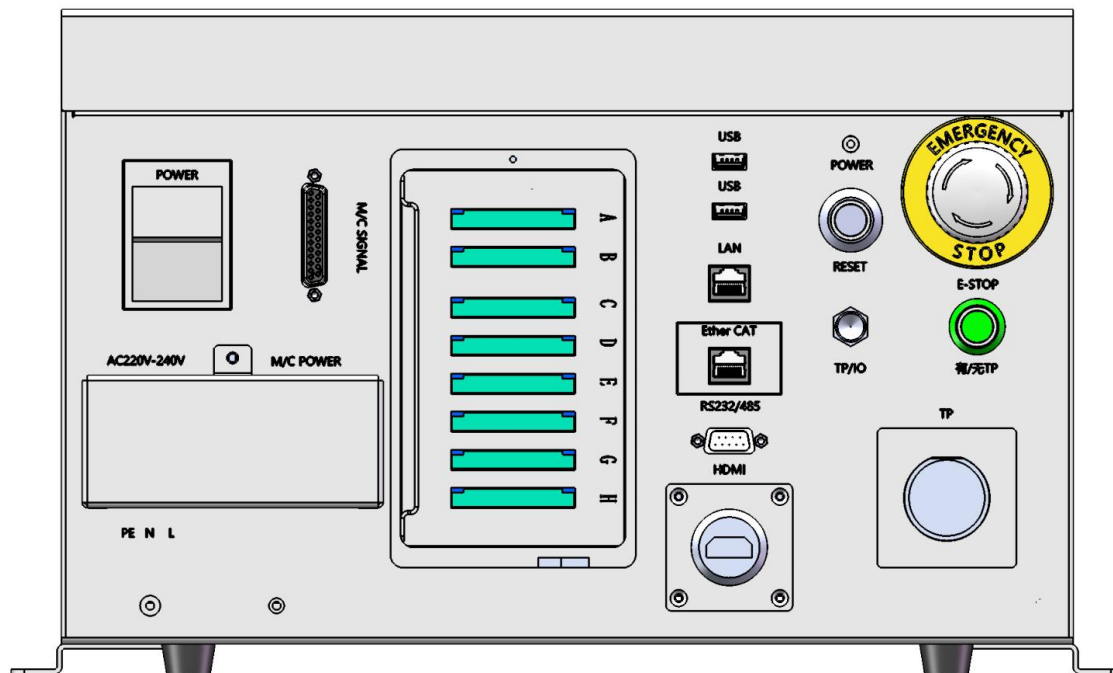
In robot parameter configuration, Gorilla Studio supports robot settings, simulation settings, external axis settings, motion parameter settings, workspace constraints, zero-point settings, and axis parameter settings.

In status monitoring, local and global variable monitoring is supported. The robot's current TCP pose, axis positions, and external axis angles can be monitored. IO monitoring, safety door monitoring, and multi-task monitoring are also supported.

1.2. Operating Environment

Terminal Type	Tablet Computer	Computer
Memory	4GB	4GB
Hard Disk Capacity	32GB	32GB
Network Communication	Wi-Fi	Wi-Fi or wired network card
Operating System	Android 8 and above	Windows 7 and above, Linux distribution
Processor	Kirin 695 or Snapdragon 660 and above	Intel Core I3 and above, ARM CortexTM-A53 and above

1.3. Connecting to the Network Port



The diagram above shows the electric cabinet. First, connect one end of the network cable to the LAN port and the other end to the terminal. Then, set the IP segment of the laptop's network port to 192.168.100.XX. Open the web page and enter 192.168.100.50:8081 to enter the page.

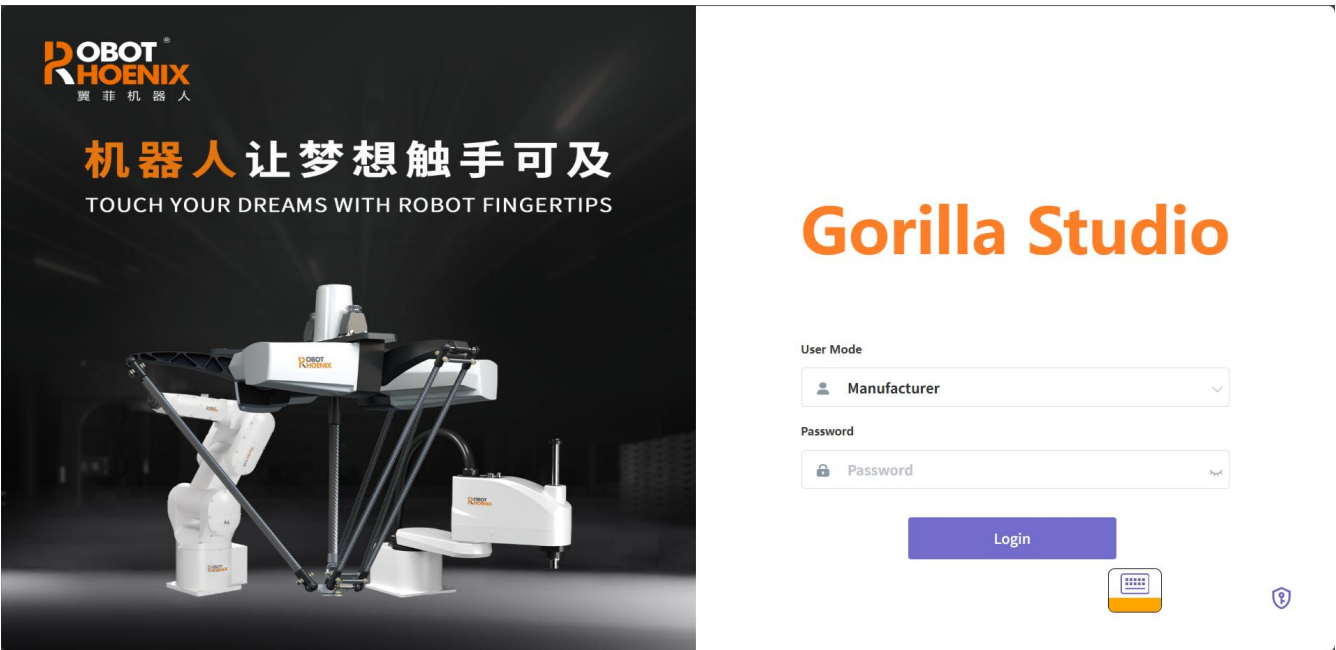
1.4. Login

Gorilla Studio system supports running on desktop browsers, but in order to improve your user experience, it is recommended to use the Chrome browser, which has been factory tested and verified and can be used with the robot controller.

Operation Instructions

To access the Gorilla Studio webpage, enter 192.168.100.50:8081 in the browser to jump to the login page.

To log in to the system, you need to enter the authorization code. Click  at the bottom right corner of the login page, copy or download the serial number, and send it to the manufacturer to obtain the authorization code. Then paste or upload the authorization code. Select your username, enter your password, and you can log in, as shown below:



The robot system has a factory-set username and initial password. It is recommended that you change the password and save it after logging in.

The username and initial password are as follows:

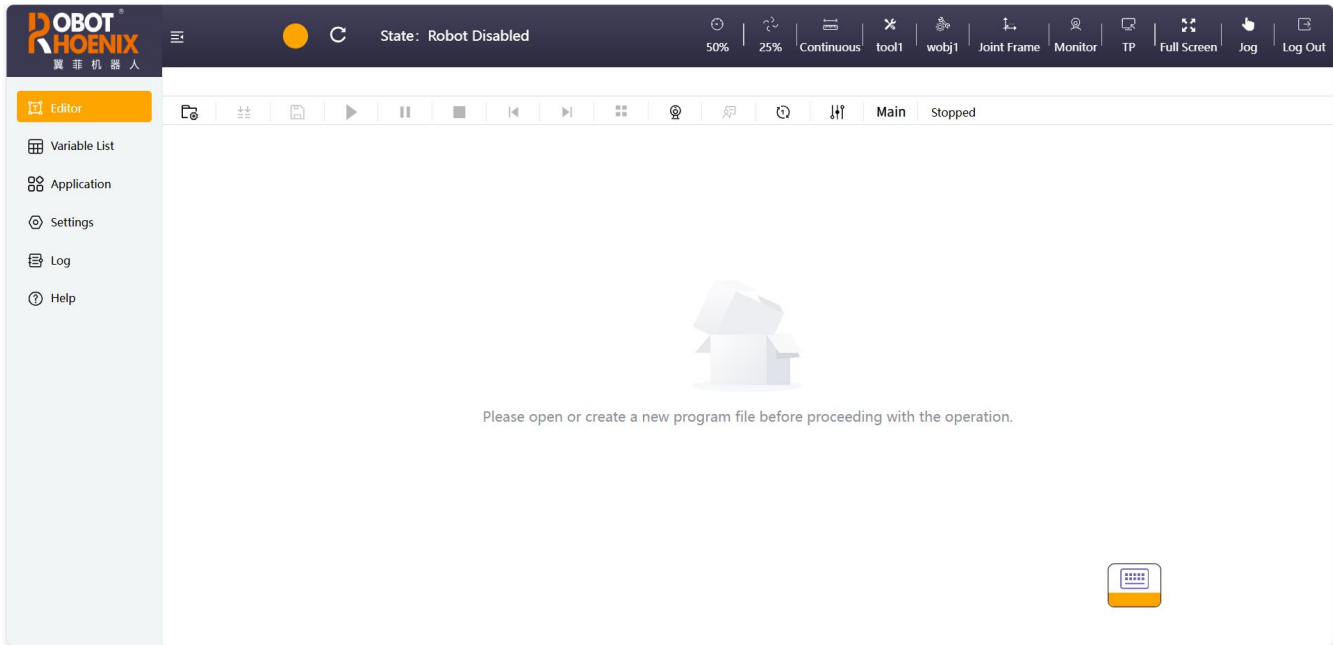
Username	Password
----------	----------

Administrator	123456
Programmer	123456
Operator	123456

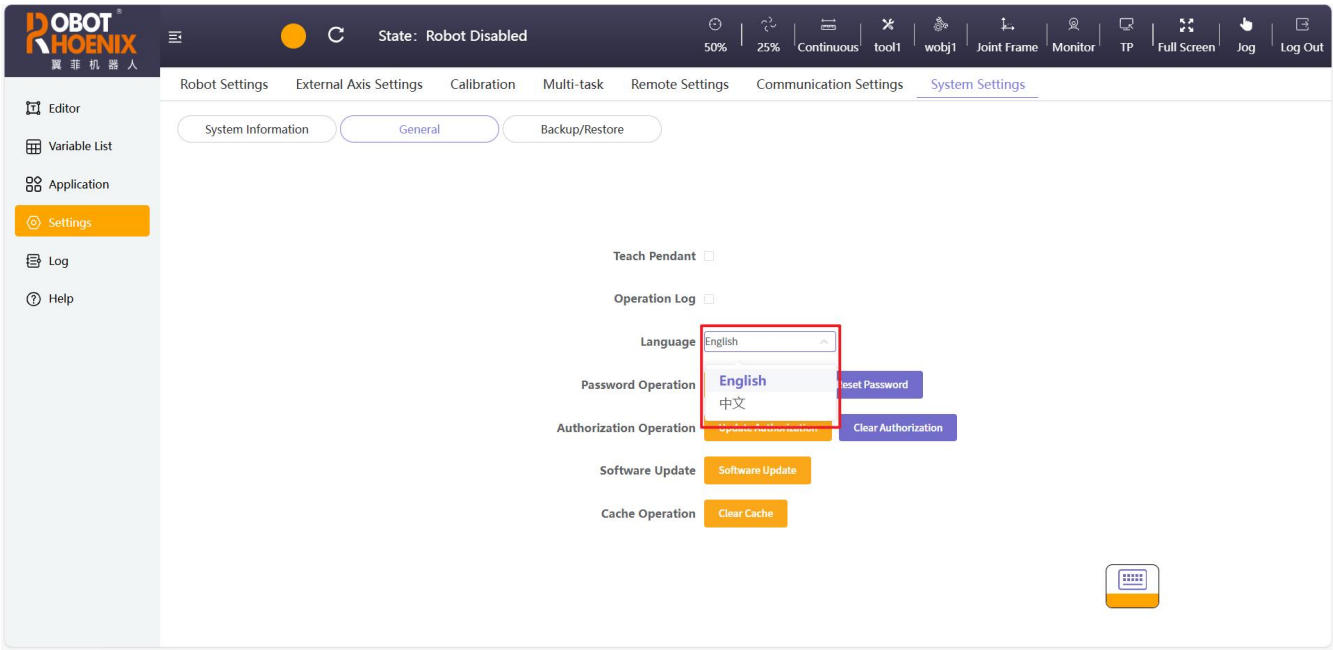
1.5. Interface

1.5.1. Introduction to the Main Interface

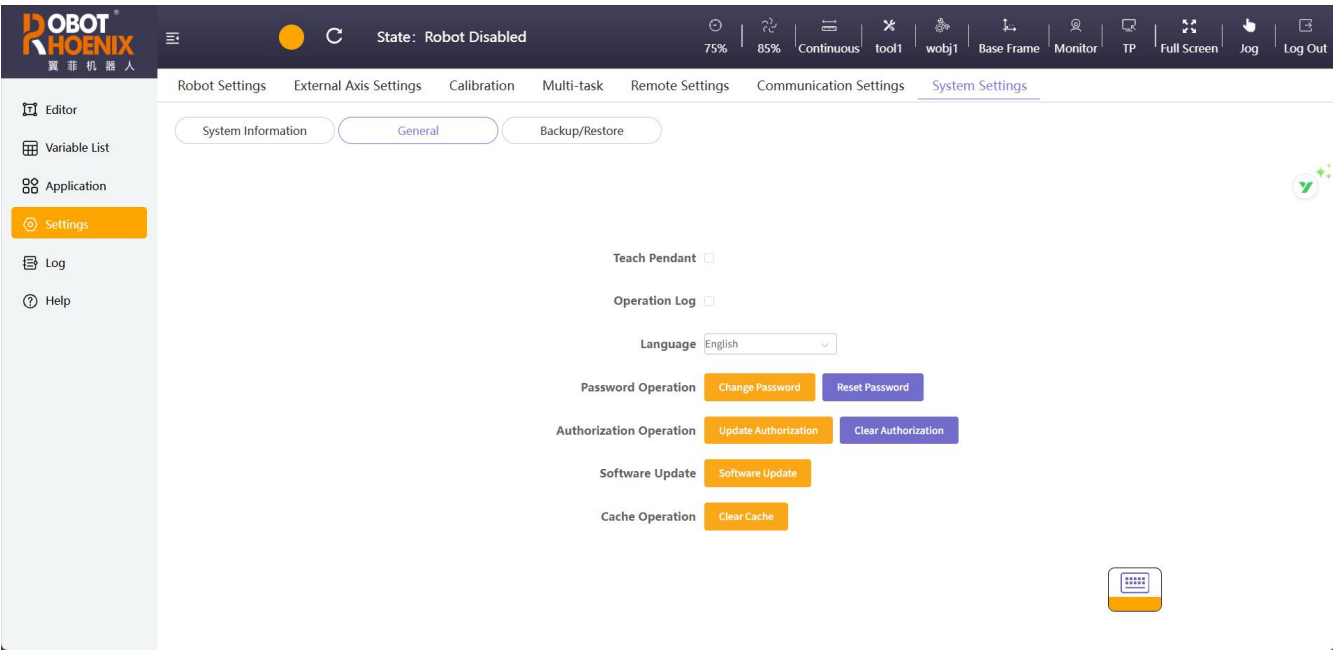
After logging in, the main interface is shown in the diagram below.



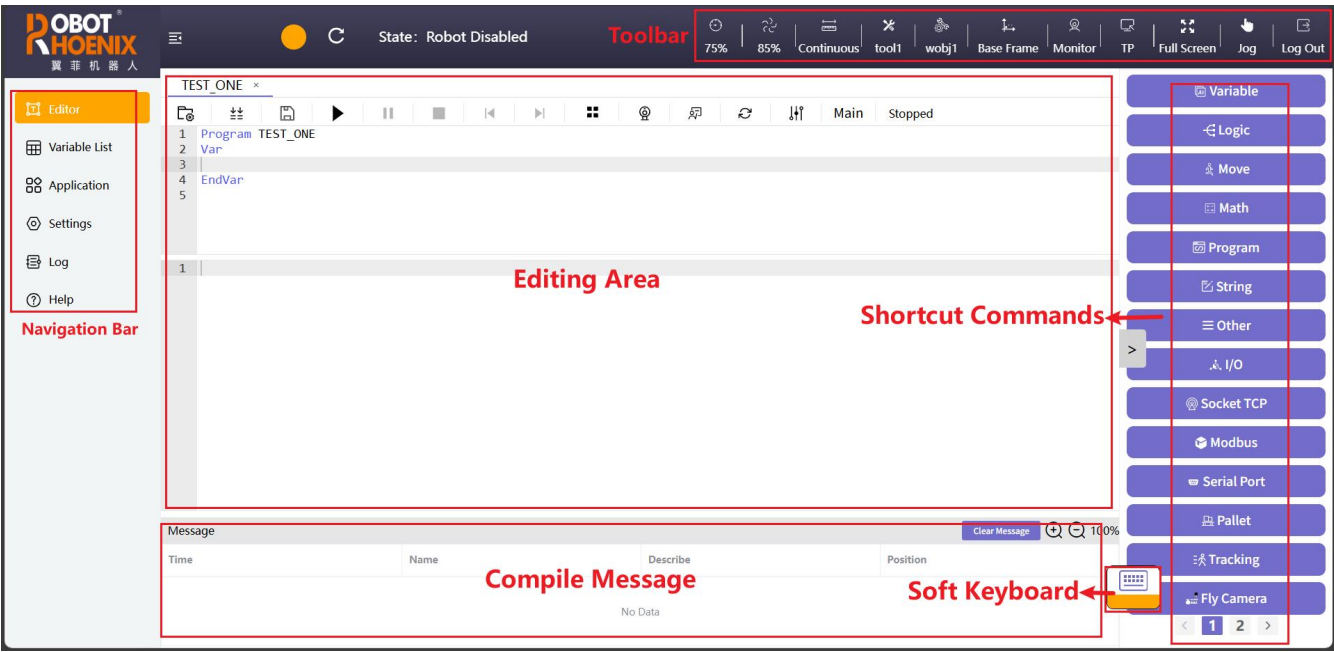
You can change the system language via “Settings” → “System Settings” → “General” → “Language”, supporting both Chinese and English, as shown below:



After switching to English, the display will appear as shown below:



Create a new program “Program 1” and open it. The interface is shown below:



The interface consists of the following sections:

- (1) Navigation Bar
- (2) Toolbar
- (3) Editing Area
- (4) Shortcut Commands
- (5) Compile Messages
- (6) Soft Keyboard

1.5.2. Navigation Bar

Items	Description
Editor	Allows creating new programs, opening programs, creating new folders, uploading program files, downloading program files, etc.
Variable table	Allows adding global variables, uploading global variables, downloading global variables, viewing tool coordinate system, viewing work object coordinate system, viewing user coordinate systems, viewing or modifying speed setting table, viewing or modifying acceleration setting table, viewing or modifying turning zone dimension setting table.
Application	Allows uploading plugins and configuring plugin extensions

Settings	Allows setting robot basic information, turning on or off simulation mode, configuring external axes, performing coordinate system calibration and zero point calibration, activating multitasking, switching control devices, setting communication configurations, setting system information, and setting extension plugins.
Log	Allows viewing error messages, warning messages and other information from the master computer and slave computer.
Help	Allows viewing error ID documents and software manuals.

1.5.3. Toolbar

Icons	Items	Description
  	Robot status	Red: Robot failure/robot emergency stop Green: Robot enabled Yellow: Robot not enabled
	Reset	Reset robot status
	Robot status	Current robot status
	Jog rate	Robot jog rate, ranging from 1% - 100%
	Automatic rate	Program's running rate, ranging from 1% to 100%. Low-speed or high-speed mode can be selected.
	Jog Mode	Continuous Mode Robot joint jog angle (unit: °)

		Robot Cartesian jog distance (unit: mm)
 tool1	Tool coordinate system	Tool coordinate system tool1 - tool16
 wobj1	Work object coordinate system	Work object coordinate system wobj1 - wobj16
 关节坐标系	Jog coordinate system	Reference coordinate system for robot jog: Joint coordinate system / base coordinate system / tool coordinate system / user coordinate system / work object coordinate system
 监控	Monitoring	Open/Close monitoring page: Allows viewing the robot's TCP position, external axis angles, I/O, safety doors, enabled tasks.
 TP	Control Mode	Switches the control device, e.g., Teach Pendant / IO / Modbus / PLC
 全屏	Full screen	Full screen or exit full screen
 点动	Jog	Robot jog/external axis jog/point to point
 退出	Log out	Log out

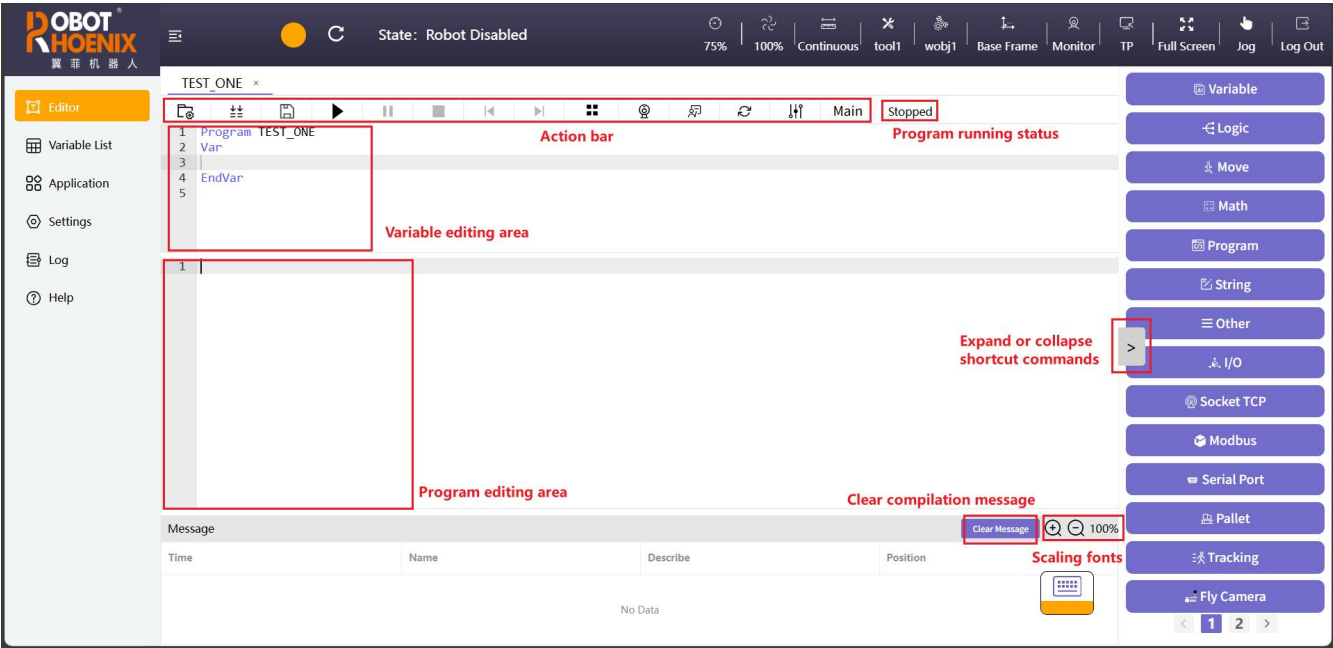
Note:

The jog coordinate system ( 关节坐标系) is not only the reference coordinate system for the robot's jog, but also the reference coordinate system for the toolbar - "Monitoring" - "Position" and the reference coordinate system when teaching the RobotTarget variable.

When the jog interface is opened, successfully compiling or running a program file will collapse the jog interface.

1.5.4. Editing Area

The editing area is mainly composed of the program operation bar, variable editing area, and program editing area, as shown in the diagram below.



Program operation bar:

Items	Description
	File management: you can create new folders, create new programs, copy files, paste files, delete files, rename files, upload files, and download files.
	Compile program
	Save program
	Start program/continue program
	Pause program
	Stop program
	Execute the previous program
	Execute the next program
	Line operation (move line up, move line down, delete line, copy, paste, comment)

	line, uncomment, undo)
	Variable monitoring. Monitor and modify local variables and global variables.
	Teaching the current point, only valid for the “JointTarget” variable and “RobotTarget” variable in the variable editing area.
	Program operation mode, loop execution or single execution
	Point Debugging: Modify the values of local or global points.
Main	Monitor the running status and local variables of different tasks.

Note:

- (1) The “Backup” folder contains deleted program files. When “Settings” → “Remote Settings” → “Select Control Device” is set to Teach Pendant, you can copy or delete program files in the Backup folder. When the control device is set to IO, Modbus, or PLC, program files in the Backup folder can only be read, and no other operations are allowed.
- (2) The “Example” folder contains sample programs. When “Settings” → “Remote Settings” → “Select Control Device” is set to Teach Pendant, you can copy program files in the Example folder. When the control device is set to IO, Modbus, or PLC, program files in the Example folder can only be read, and no other operations are allowed.

1.5.5. Shortcut Commands

Items	Description
Variable	Declare local variables, including basic type variables and complex type variables
Logic commands	Insert logic commands, If/For/While/Goto/Exit/Continue/Not/And/Xor/Or/Wait/Pause/Resume/Case
Motion commands	Insert motion commands, MoveL/MoveJ/MoveC/MoveAbsJ/MoveLJump/MovePath/AutoLJM
Mathematical commands	Insert mathematical operation commands, Sin/Cos/Tan/Asin/Acos/Atan/Exp/Ln/Log10/Pow/Sqrt/Ceil/Floor/Round/Min/Max/Abs/Sign
I/O	Insert I/O commands, SetDO/GetDI/PulseDO/Group/SetOG/GetIG
Program	Insert call subprogram command, Call

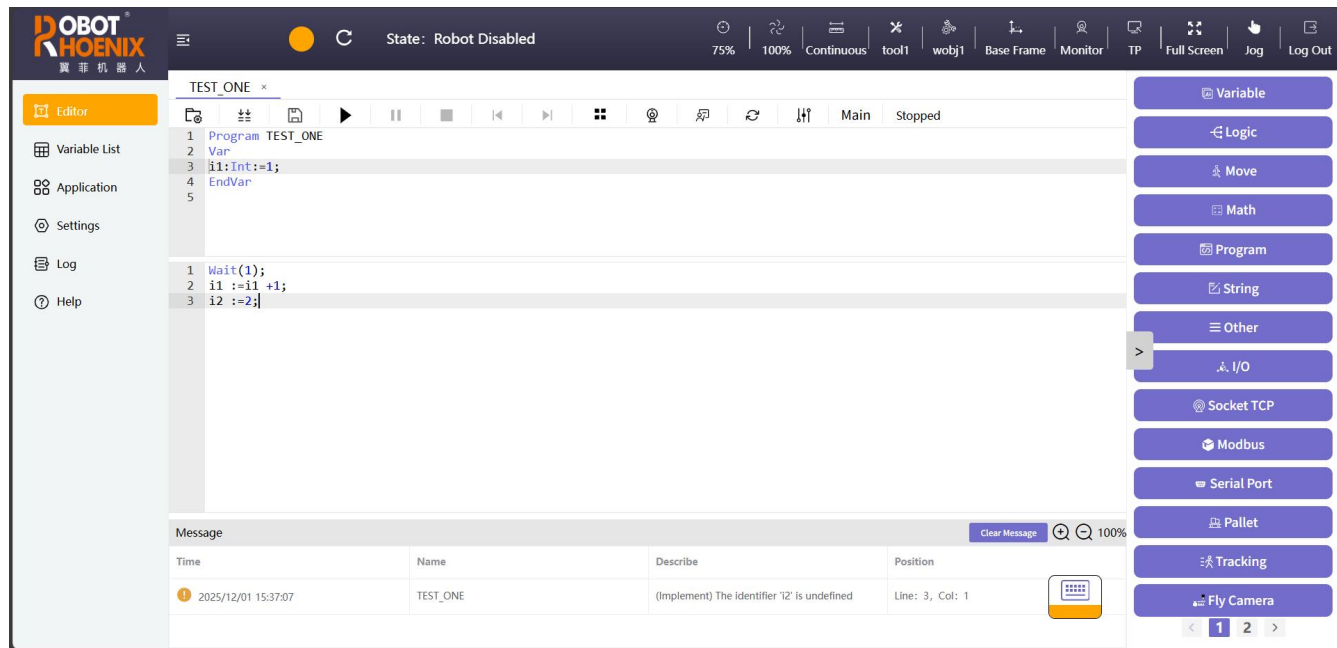
SocketTCP	Insert SocketTCP commands, SocketCreate/SocketClose/SocketSendString/SocketReadString/SocketStatus
String	Insert string commands, StrConcat/StrDelete/StrFind/StrInsert/StrLeft/StrLen/StrMid/StrReplace/StrRight/StrToDoubler/StrToInt/StrToWord/StrToFloat/DoubleToStr/DoubleToInt/DoubleToWord/DoubleToFloat
Other	Insert other commands, GetCartesianPose/GetAxisPosition/GetRobotTarget/GetRobotStatus/GetRunStatus/ZeroCalibration/UToEuler/EulerToU/BitSet/BitCheck/PoseMultiplication/PoseInverse/TimerStart/TimerOut/Print
Pallet	Insert pallet command, generate pallet variable, get pallet point, SetPallet/SetPallet4/Pallet/PalletSequence
Modbus	Insert Modbus commands, GetModbusCoilToBool/SetModbusCoilByBool/GetModbusRegToWord/SetModbusRegByWord/GetModbusRegToInt/SetModbusRegByInt/GetModbusRegToDouble/SetModbusRegByDouble/GetModbusRegToFloat/SetModbusRegByFloat/ModbusTcpStatus/GetModbusCoil/SetModbusCoil/GetModbusReg/SetModbusReg
Track	Insert track commands, ConveyorVision/ConveyorSensor/GetConveyorObject/TrackBase/TrackConveyor
Fly camera	Insert fly camera commands InitialFly/SetFly/StartFly/StopFly/CheckFly/GetFlyRobotPose/CalculationFlyResult

1.5.6. Compile Information

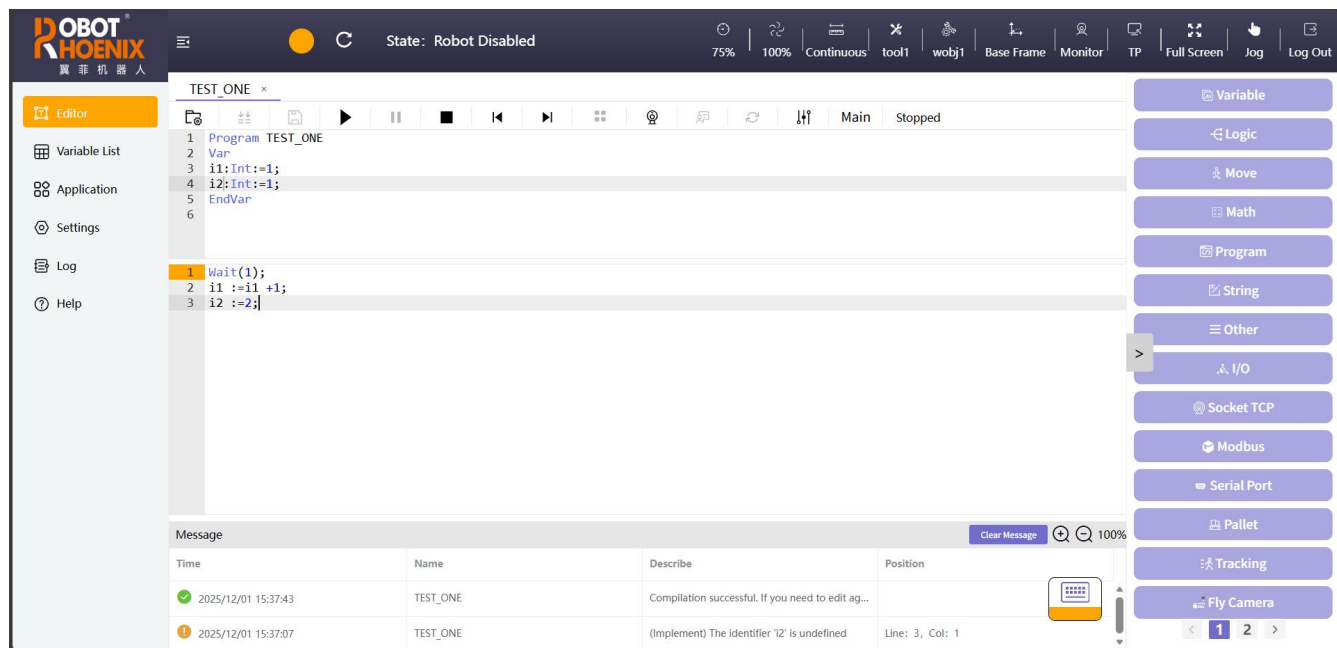
Operation method

Click the “Compile” () or “Start/Continue” () button in the program operation bar to compile or run the program file. The problem message area will display the compilation

success information or error information. As shown below:



Compilation Error: In the program editing area, line 3, column 1, “i2” is undefined. You must correct it and compile successfully before the program can run.

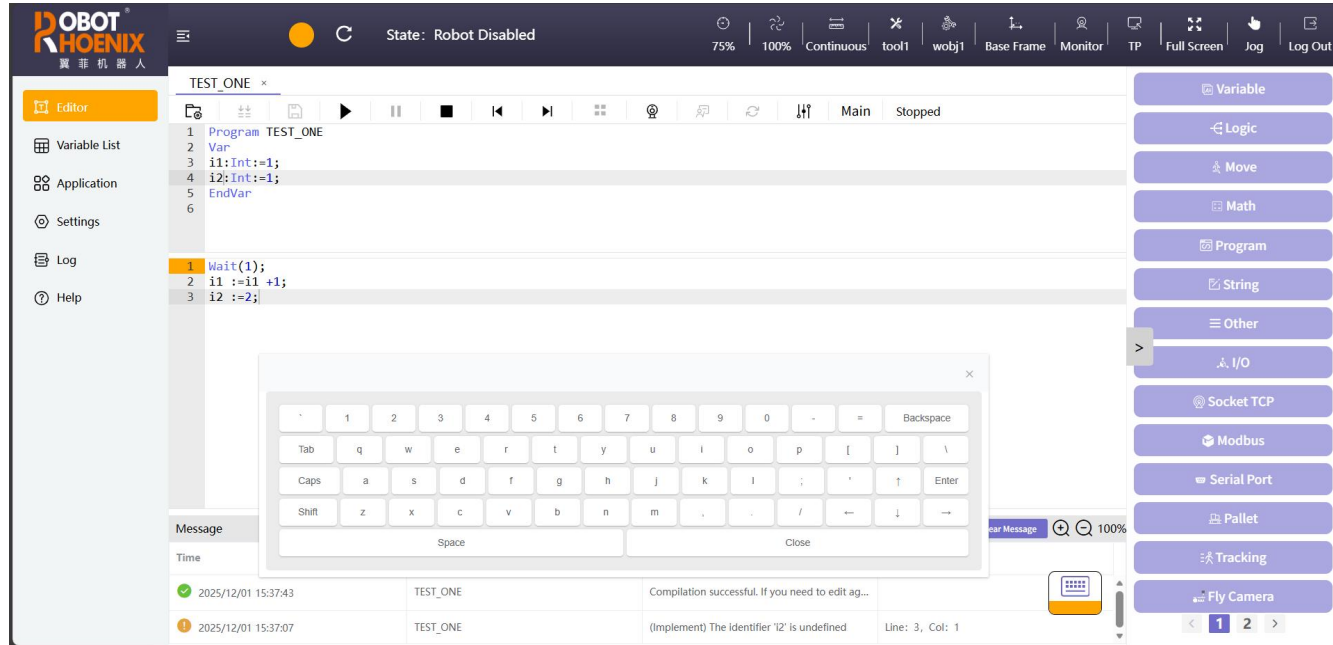


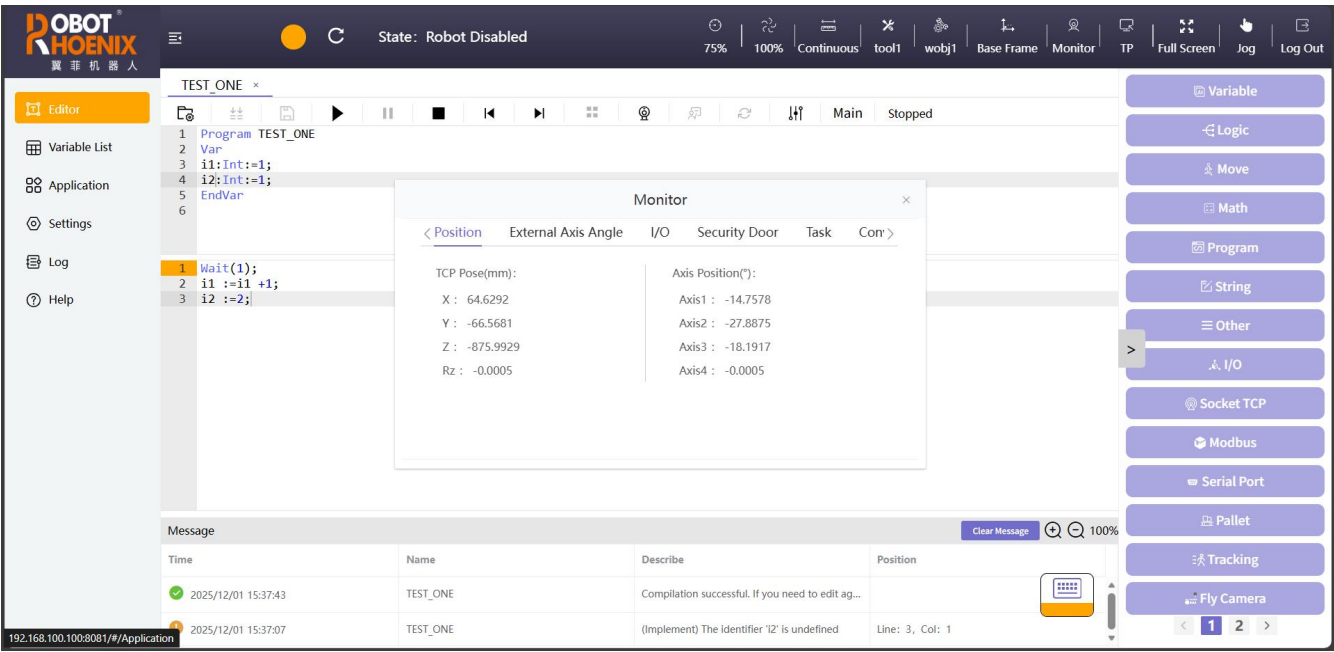
After the modification is correct and the compilation is successful, the program can be run.

1.5.7. Virtual Keyboard

Operation method

Click the virtual keyboard icon  to open the virtual keyboard for text input. Click it again to close the keyboard. Long press the left mouse button on the virtual keyboard icon to move the icon. The virtual keyboard is opened as shown below.



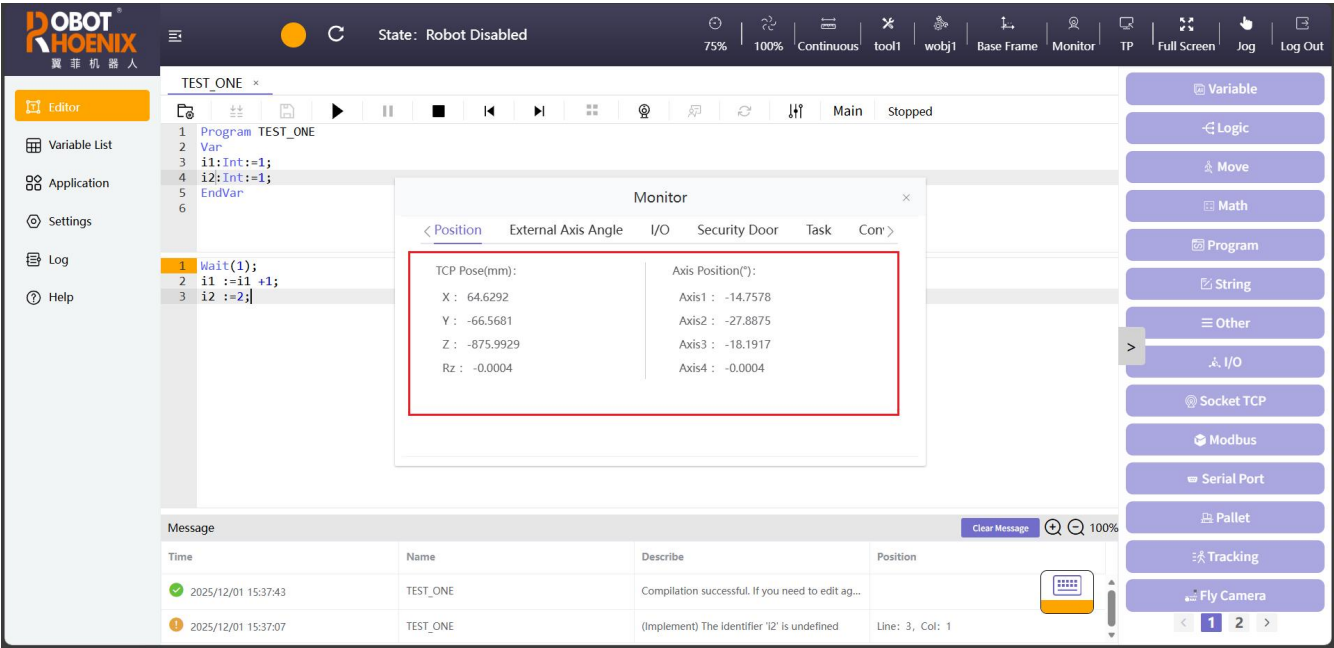


1.5.8.2. Position

Description

You can view the position of each axis in real-time (unit: °).

By clicking the “Jog Coordinate System” () on the toolbar and selecting different coordinate systems, you can view the pose of the robot TCP in the specified reference coordinate system (unit: mm). As shown below:



Operation method

- **When the base coordinate system is selected**

The coordinate value displayed in the monitoring interface “Position”-“TCP Pose” is the position and orientation of TCP (tool center point) relative to the base coordinate system.

- **When the tool coordinate system is selected**

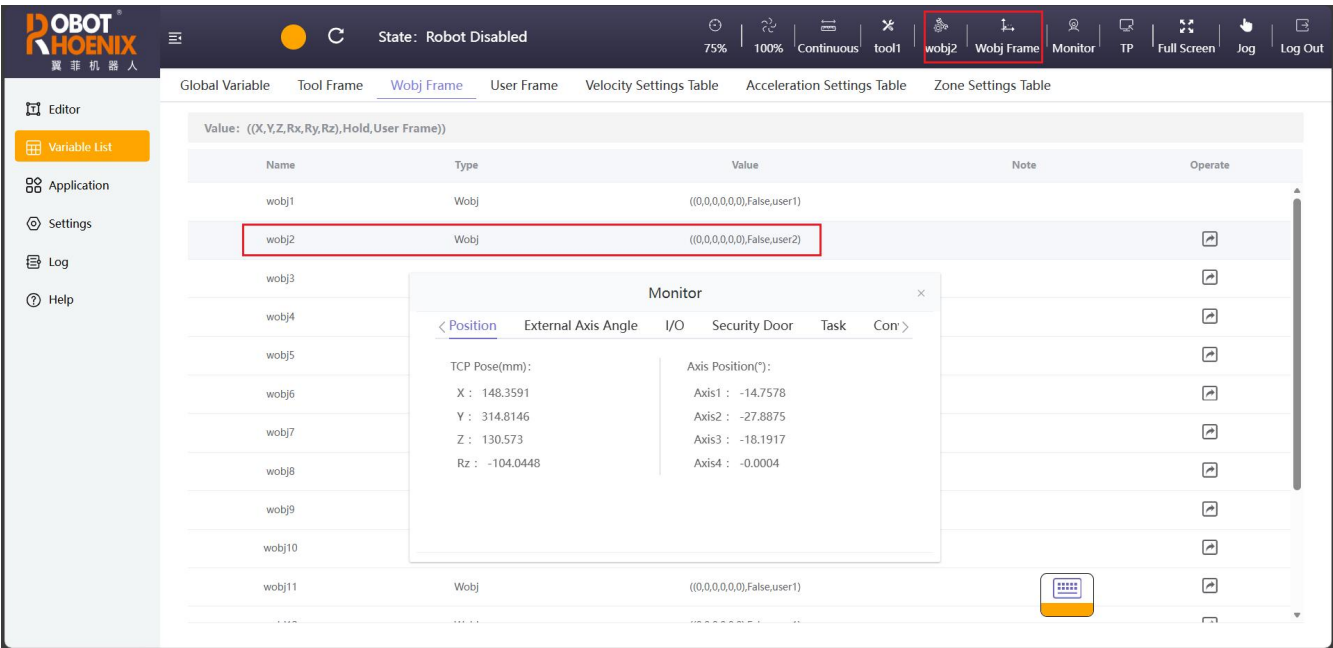
The coordinate value displayed in the monitoring interface “Position”-“TCP Pose” is the position and orientation of TCP (tool center point) relative to the base coordinate system.

- **When the user coordinate system is selected**

The coordinate value displayed in the monitoring interface “Position”-“TCP Pose” is the position and orientation of TCP (tool center point) relative to the user coordinate system. This user coordinate system is the reference user coordinate system in the work object coordinate system number selected in the toolbar “Work Object Coordinate System” (wobj3), which can be viewed through the “Variable Table”- “Work Object Coordinate System” interface.

Example:

As shown in the figure, when “wobj2” is selected in the toolbar under Work Object, you can see from the Variable Table – Work Object interface that the reference user frame of “wobj2” is “user2.” Therefore, the coordinates displayed in the monitoring interface Position – TCP Pose represent the position and orientation of the TCP (Tool Center Point) relative to “user2.” “user2” can be viewed in the Variable Table – User Frame interface.



- **When the work object coordinate system is selected**

The coordinate value displayed in the monitoring interface “Position” - “TCP Pose” is the position and orientation of the TCP (tool center point) relative to the work object coordinate system. This work object coordinate system is the work object coordinate system selected by

the toolbar “Work Object Coordinate System” ().

1.5.8.3. External Axis Angle

Description

The external axis angle (unit: °) can be viewed in real-time, as shown below:



1.5.8.4. I/O

Description

The DI/DO status information can be viewed, annotations can be added, DO set operations can be performed, or DI signals can be forcefully written.

Operation method

- **DO set operation**

Click on “I/O” - “DO” to open the DO monitoring interface. Then, click the corresponding set operation button for the DO you wish to set. As shown below:

Monitor					
< Position External Axis Angle I/O Security Door Task Con' >					
DI DO					
DO	Type	State	Operate		Note
DO1	Bool		True	False	
DO2	Bool		True	False	
DO3	Bool		True	False	
DO4	Bool		True	False	
< 1 2 3 4 5 6 ... 16 >					

Note:

If the remote control function is set for some DOs in remote IO mode, the set operation cannot be performed on them. As shown below, DO2, DO3, and DO4 are all set with the remote control function. In the monitoring interface, their set buttons are disabled and the set operation cannot be performed.

The screenshot displays the ROBOT PHOENIX software interface. The top navigation bar includes 'Robot Settings', 'External Axis Settings', 'Calibration', 'Multi-task', 'Remote Settings', 'Communication Settings', and 'System Settings'. The 'Remote Settings' window is active, showing a table with columns 'Name', 'Remote Operation', and 'Operate'. The table lists DO1 through DO10. DO2, DO3, and DO4 are highlighted with a red box, indicating they are set with remote control functions: 'Program Running', 'Program Pause', and 'Program Error' respectively. Their 'Operate' buttons are disabled. The 'Monitor' window is also open, showing a table with columns 'DO', 'Type', 'State', 'Operate', and 'Note'. The table lists DO1 through DO4. DO2, DO3, and DO4 are highlighted with a red box, indicating they are set with remote control functions: 'True', 'True', and 'True' respectively. Their 'Operate' buttons are disabled.

- Forcefully write DI signal:

Click "I/O" - "DI" to open the DI monitoring interface, check the "Forced Signal" column of the DI to be written, and click the set operation button. As shown below:

Monitor

< Position

External Axis Angle

I/O

Security Door

Task

Con>

DI

DO

DI	Type	State	Operate		Mandatory Signal	Note
DI1	Bool		True	False	☒	
DI2	Bool		True	False	☐	
DI3	Bool		True	False	☐	
DI4	Bool		True	False	☐	

<

1

2

3

4

5

6

...

16

>

1.5.8.5. Safety Door

Description

Green: Safety door closed. Yellow: Safety door open. As shown below:

Monitor

< Position

External Axis Angle

I/O

Security Door

Task

Con>

Security Door 1	
Security Door 2	

Monitor

< Position

External Axis Angle

I/O

Security Door

Task

Con>

Security Door 1	
Security Door 2	

Note:

You can choose whether to enable the safety door function, and it will be configured before delivery according to the user's requirements.

- Enable the safety door:

(1) When the teach pendant is not enabled, the program can only be started when both safety doors 1 and 2 are closed (green);

(2) When the teach pendant is enabled, the program can only be started when both safety doors 1 and 2 are closed (green) in automatic mode.

(3) When the teach pendant is enabled, the program can be started in manual mode regardless of whether the safety door is closed.

- Disable the safety door:

The program can be started regardless of whether the safety door is closed.

1.5.8.6. Tasks

Description

View the activated multi-tasks. As shown below:

Monitor				
< Position External Axis Angle I/O Security Door <u>Task</u> Con' >				
Task	Program	Task Type	Running State	Active State
1	Program	Static	Stop	Not Active
2		Non-static	Stop	Not Active
3		Non-static	Stop	Not Active
4		Non-static	Stop	Not Active

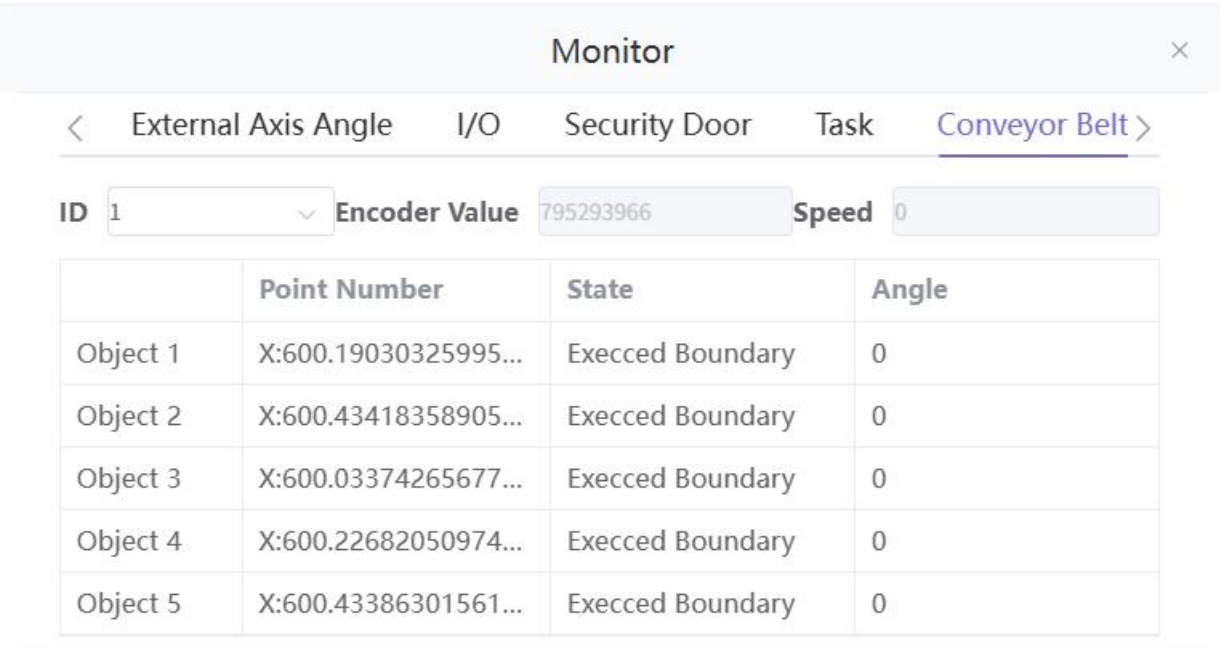
1.5.8.7. Conveyor Belt

Description

The BeltTracking plugin must be installed.

You can view information related to the conveyor belt, including the first five objects on each of the four conveyor belts, as well as the current conveyor encoder value and conveyor speed.

Object information includes coordinates, object status, and the angle of the object on the rotary conveyor. As shown in the figure below:



1.6. Basic Terminology

1.6.1. Model

Name	Type	Number of axes	Series/Parallel	Features	Applications
Six-axis robot		6	Series	Highly flexible, suitable for working in almost any trajectory or angle.	Used in manufacturing, automated assembly, welding, cutting, handling and other

					occasions that require precise control and repetitive operations.
SCARA robot		4	Series	Lightweight structure and quick response.	Mechanical assembly, material delivery, distribution and other occasions. Robot assembly, labeling, patching, dispensing, etc. in the 3C industry.
Delta robot		4	Parallel	High precision and fast speed.	Handling and sorting of medicines and food, etc.

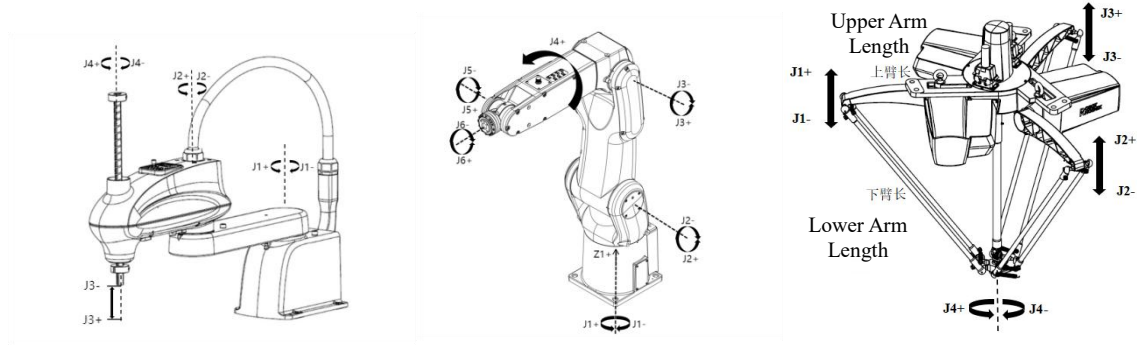
1.6.2. Coordinate System

In the Gorilla Studio robot control system, there are five coordinate systems: joint coordinate system, base coordinate system, tool coordinate system, user coordinate system, and work object coordinate system.

1.6.2.1. Joint Coordinate System

Description

The joint coordinate system exists at each joint of the robot.

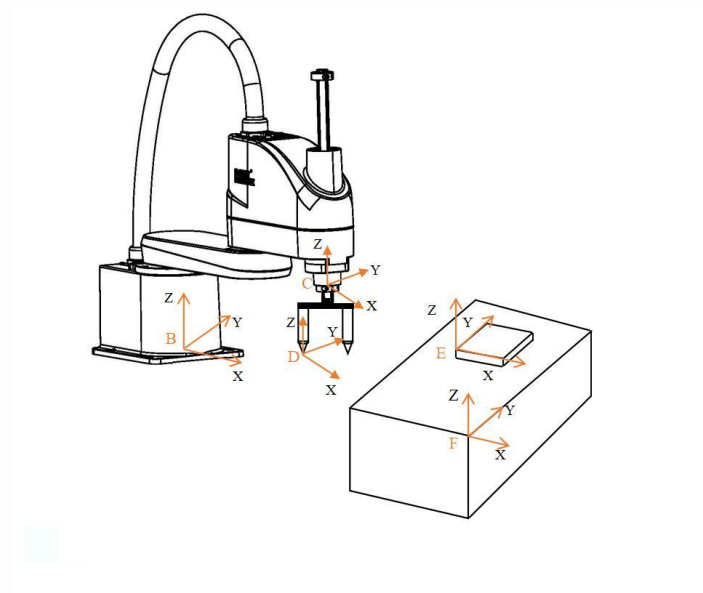


1.6.2.2. Base Coordinate System

Description

The base coordinate system is generally located at the root of the robot and is used to determine the robot's placement position.

As shown in the figure below, B is the base coordinate system, C is the flange coordinate system, D is the tool coordinate system, E is the work object coordinate system, and F is the user coordinate system:

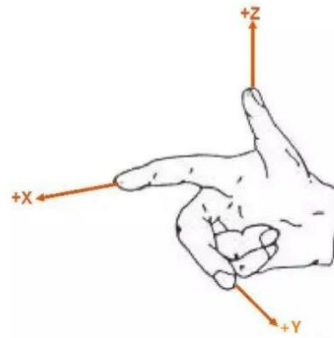


Note:

The base coordinate system is a Cartesian coordinate system (the tool coordinate system, user coordinate system, and work object coordinate system are also Cartesian coordinate systems), and its values are expressed using ZYX Euler angles.

The method for determining the direction of a Cartesian-type coordinate system is as follows:

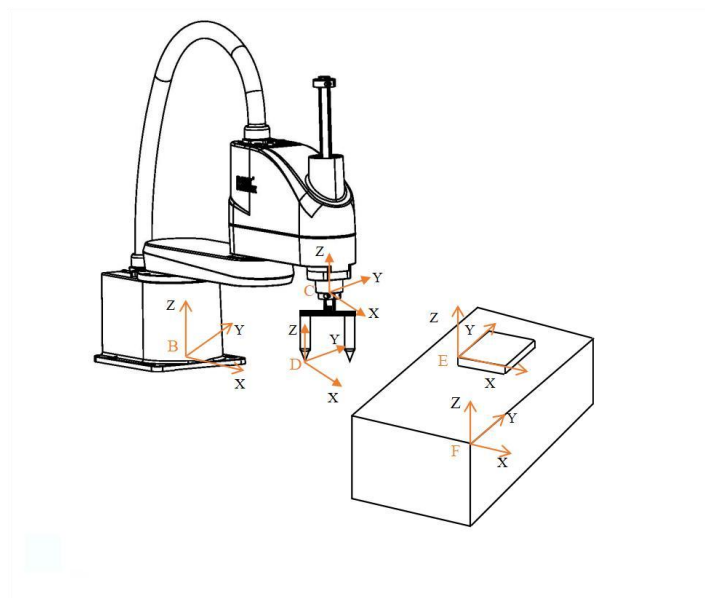
first, determine the X and Z directions, and then use the “right-hand rule” to determine the Y direction.



1.6.2.3. Tool Coordinate System

Description

The tool coordinate system is attached to the tool. TCP (Tool Center Point) is used as the reference point to measure the robot's arrival position. When using ordinary tools, the tool coordinate system is defined relative to the flange coordinate system. When using external tools, the tool coordinate system is defined relative to the base coordinate system. Generally, the tool end point is taken as TCP, and the direction can be freely defined. As shown in the figure below, B is the base coordinate system, C is the flange coordinate system, D is the tool coordinate system, E is the work object coordinate system, and F is the user coordinate system:



Note:

In the Gorilla Studio robot control system, 16 tool coordinate systems are defined. Tool1

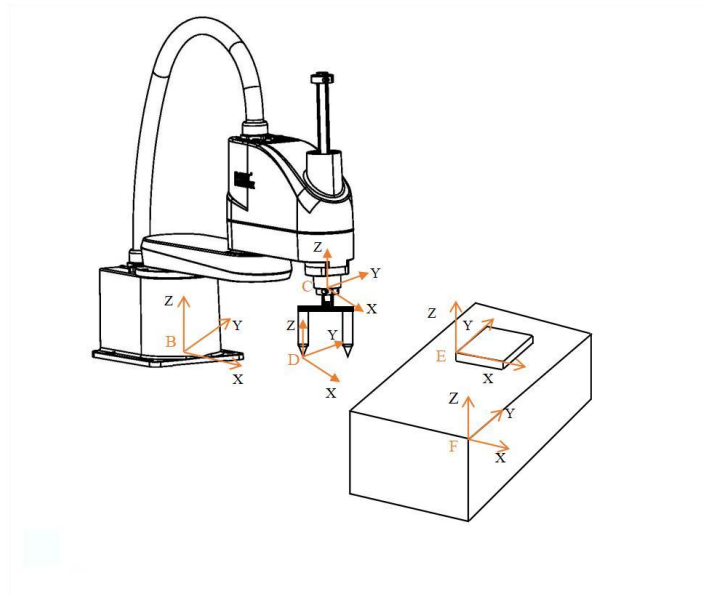
means no tool is used, and the tool coordinate system is located at the end of the robot and coincides with the flange coordinate system. Tool2-tool16 can be freely defined by the user.

1.6.2.4. User Coordinate System

Description

The user coordinate system is a user-defined coordinate system that serves as the reference coordinate system of the work object coordinate system. Its origin parameters are described as coordinate values in the robot base coordinate system. The user coordinate system can be set arbitrarily in essence, but in specific applications it can be set on specific objects, such as work tables, conveyor belts, etc.

As shown below, B is the base coordinate system, C is the flange coordinate system, D is the tool coordinate system, E is the work object coordinate system, and F is the user coordinate system:



Note:

In the Gorilla Studio robot control system, 16 user coordinate systems are defined. User1 means no additional user coordinate system is used, and the user coordinate system coincides with the base coordinate system. User2-user16 can be freely defined by the user.

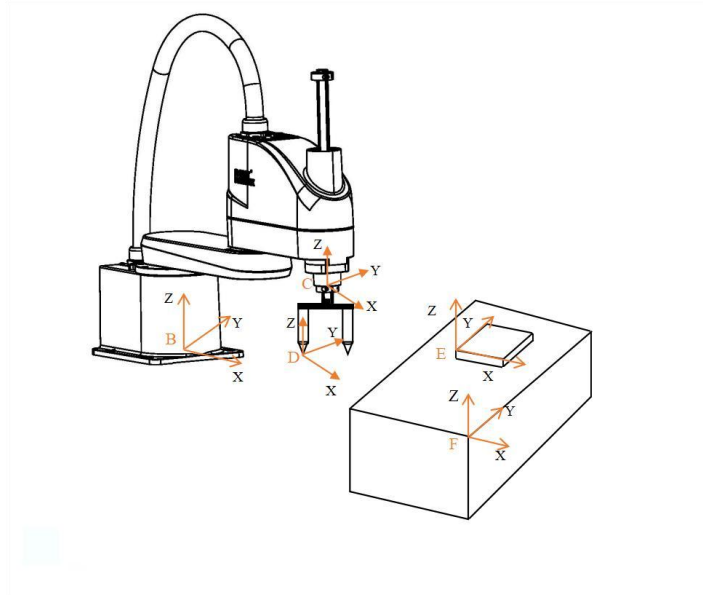
1.6.2.5. Work Object Coordinate System

Description

The work object coordinate system is the reference coordinate system used by the robot end effector to describe the position and orientation of the workpiece with which it interacts. The

positions used in all motion commands are defined in the work object coordinate system. The work object coordinate system is defined relative to the user coordinate system, and its origin parameters are described as coordinate values in the robot's user coordinate system. The work object coordinate system can be set arbitrarily in nature, but it can be set on a specific workpiece in specific applications.

As shown below, B is the base coordinate system, C is the flange coordinate system, D is the tool coordinate system, E is the work object coordinate system, and F is the user coordinate system:



Note:

In the Gorilla Studio robot control system, 16 work object coordinate systems are defined. Among them, wobj1 means that the workpiece is not used, and the work object coordinate system coincides with the user coordinate system. wobj2-wobj16 can be freely defined by the user.

1.7. Basic Robot Operations

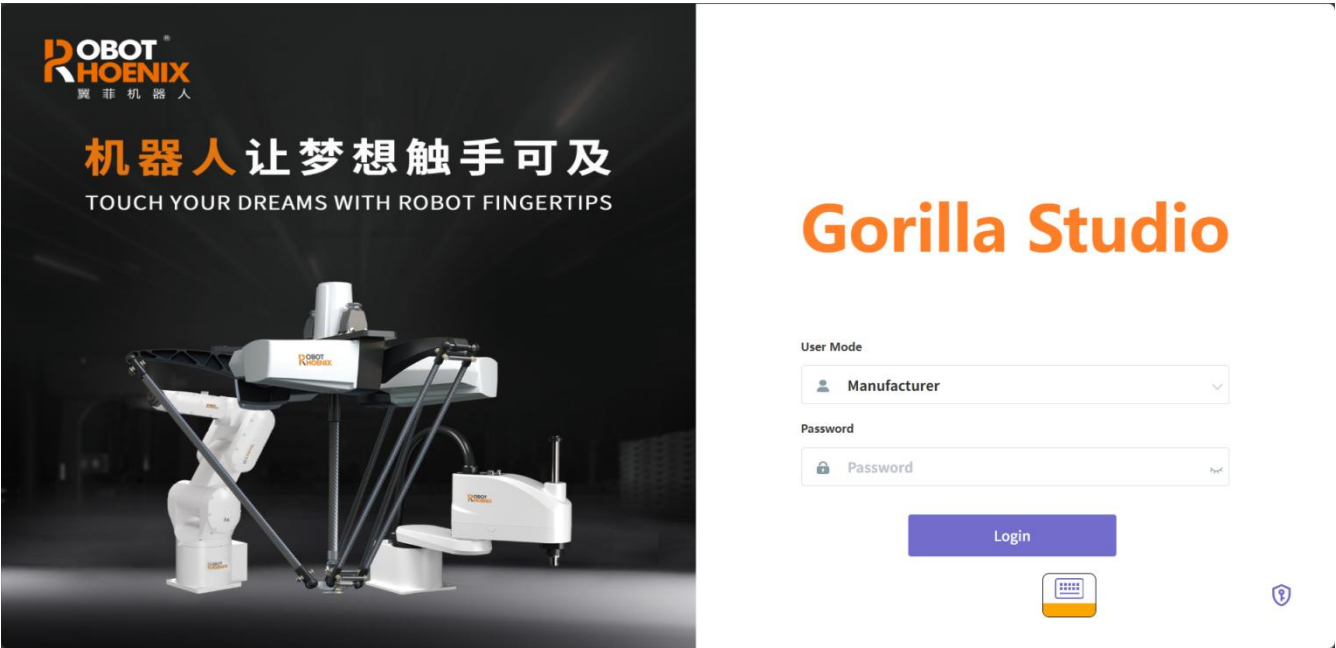
1.7.1. User Permissions Description

Gorilla Studio has four login permission levels: Operator, Programmer, Administrator, and Manufacturer, each with different access rights. The hierarchy of permissions is as follows:

Operator < Programmer < Administrator < Manufacturer.

Operation method

In the upper-right corner of the toolbar, click the “Logout” button to return to the login screen. Select the username, enter the corresponding password, and click “Login” to proceed.



Description

Different users have different operation permissions, as shown in the table below.

		Operator	Programmer	Administrator	Manufacturer
Jog	/	×	√	√	√
Rate Settings	/	×	√	√	√
Coordinate System Selection	/	√	√	√	√
Monitoring	/	View Only	√	√	√
Remote Settings	/	×	√	√	√
File Management	New Folder	×	√	√	√
	New Program				
	Copy				
	Paste				
	Delete				
	Rename				
	Upload File				

	Download File				
Program	View Program	√	√	√	√
	Edit Program	×	√	√	√
	Compile	√	√	√	√
	Start				
	Stop				
	Pause				
	Previous Step				
	Next Step				
	Variable Monitoring	View Only	√	√	√
	Run Mode	√	√	√	√
	Point Debugging	×	√	√	√
Variable Table	Global Variable Table	View Only	√	√	√
	Turning Zone Size Setting Table		√	√	√
	Velocity setting table		√	√	√
	Acceleration setting table		√	√	√
	Tool coordinate system		View Only	√	√
	Work object coordinate system			√	√
	User coordinate system			√	√
	Upload/Download Global Variables	×	√	√	√
Settings	Robot Settings	×	×	×	√
	External Axis Settings		×	×	√
	Calibration		×	√	√
	Multi-Task		√	√	√

	Remote Settings		√	√	√
	Communication Settings		√	√	√
System Settings	Language	√	√	√	√
	Password Operations	Editable, Cannot Reset			√
	Authorization Operations	Updatable, Cannot Clear			√
	Software Update	×	×	√	√
	Clear Cache	×	×	×	√
	Backup & Restore	×	×	√	√
	Plugin Management	×	×	×	√
Logs	/	View Only	√	√	√

Note:

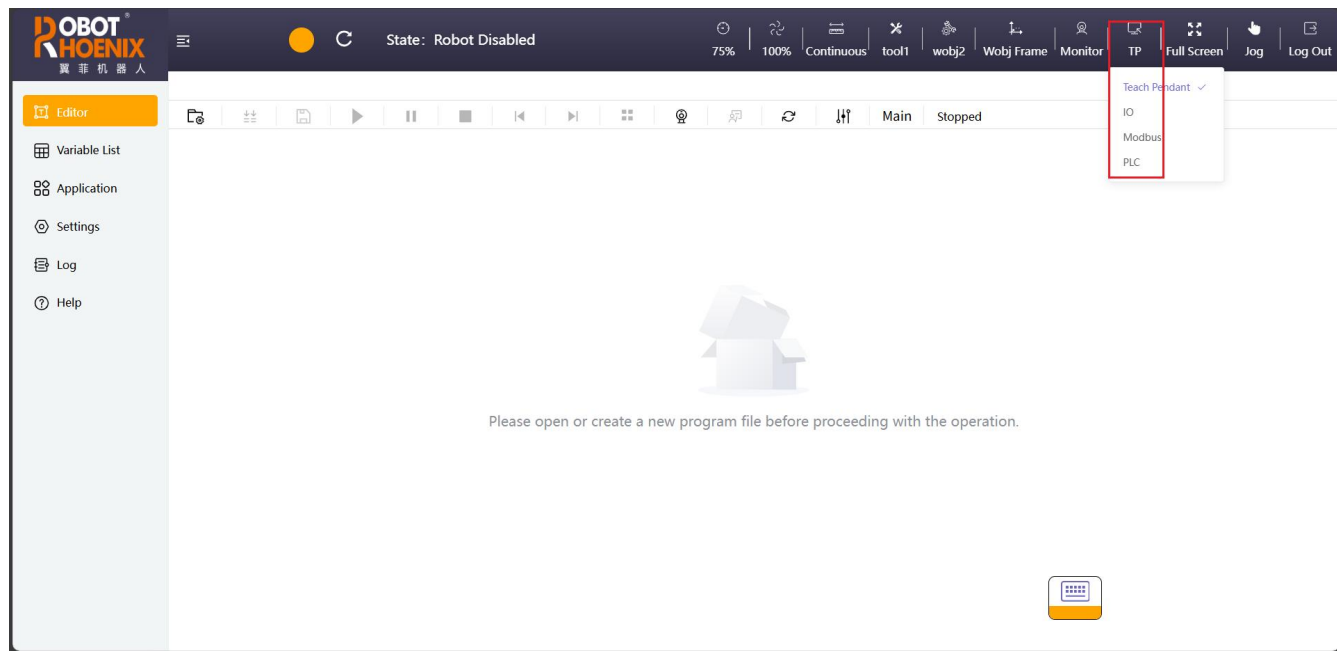
The manufacturer has the highest authority to control the software and all functions can be operated.

1.7.2. Control Device Switching

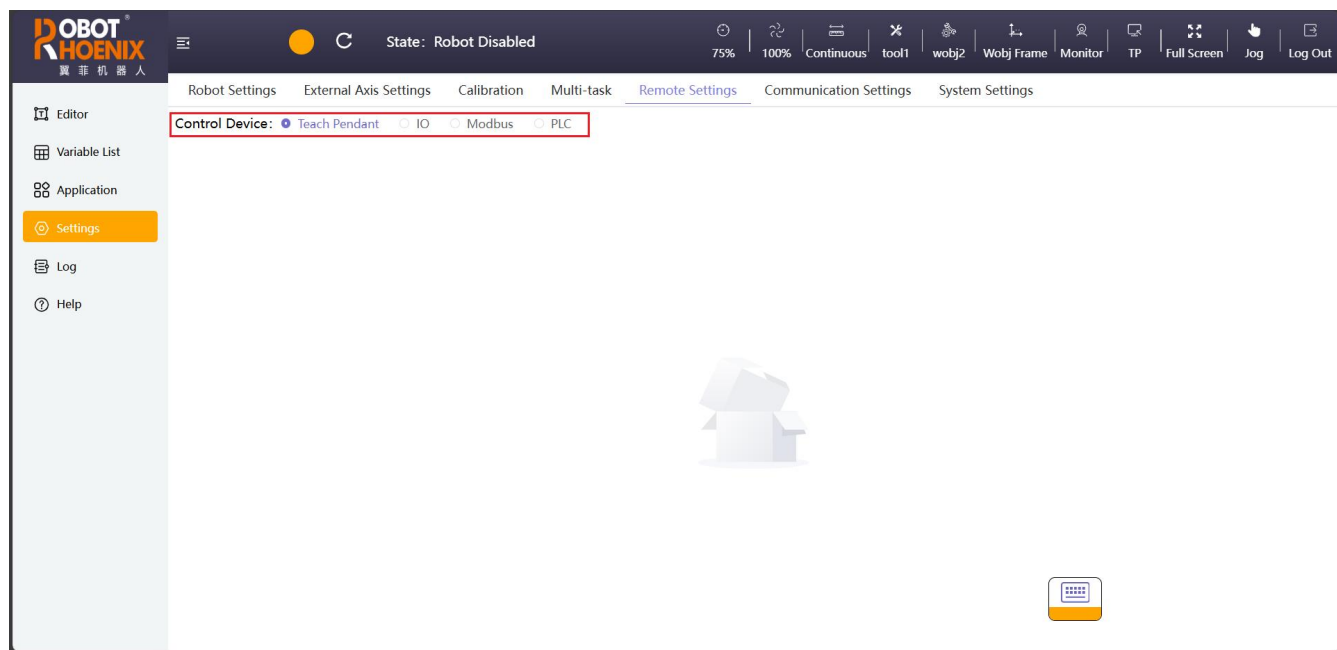
The robot controller and body can be regarded as actuators in a broad sense, and their actions are controlled by upper-level devices. The upper-level devices with control rights include the teach pendant, remote IO module, remote Modbus module and remote PLC module.

Operation method

The current control device can be displayed and switched on the toolbar:



Switch “Control Device” through the left navigation bar “Settings” - “Remote Settings”, and select the required control device. As shown in the figure below:



Note:

The switching of control devices needs to be done between the “teach pendant” and other control devices, but not between other devices. That is, remote IO control, remote Modbus control and remote PLC control cannot be switched between each other.

1.7.3. Clear Fault

Operation method

When the robot status shows “Robot Fault”, click  to attempt to clear the fault.

Operation result

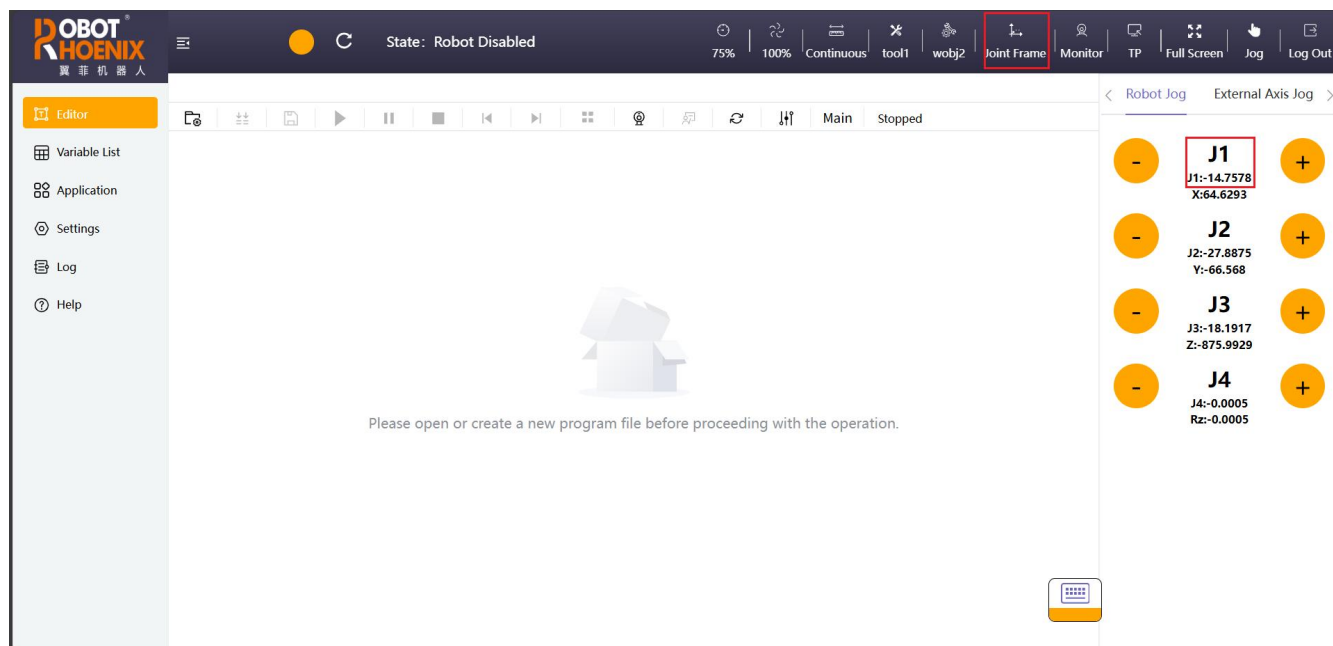
- After attempting to clear, if the issue is resolved and the fault no longer exists, the status will change to “Robot Enabled” or “Robot Not Enabled”.
- After attempting to clear, if the issue is not resolved and the fault persists, the status will remain “Robot Fault”.

1.7.4. Coordinate System Switching

Robotphoenix robots include multiple coordinate systems, among which the coordinate systems available for use in Gorilla Studio include Joint Coordinate System, Base Coordinate System, User Coordinate System, Tool Coordinate System, and Work Object Coordinate System. Switching between different coordinate systems makes it more convenient for jogging, teaching points, and viewing the TCP position.

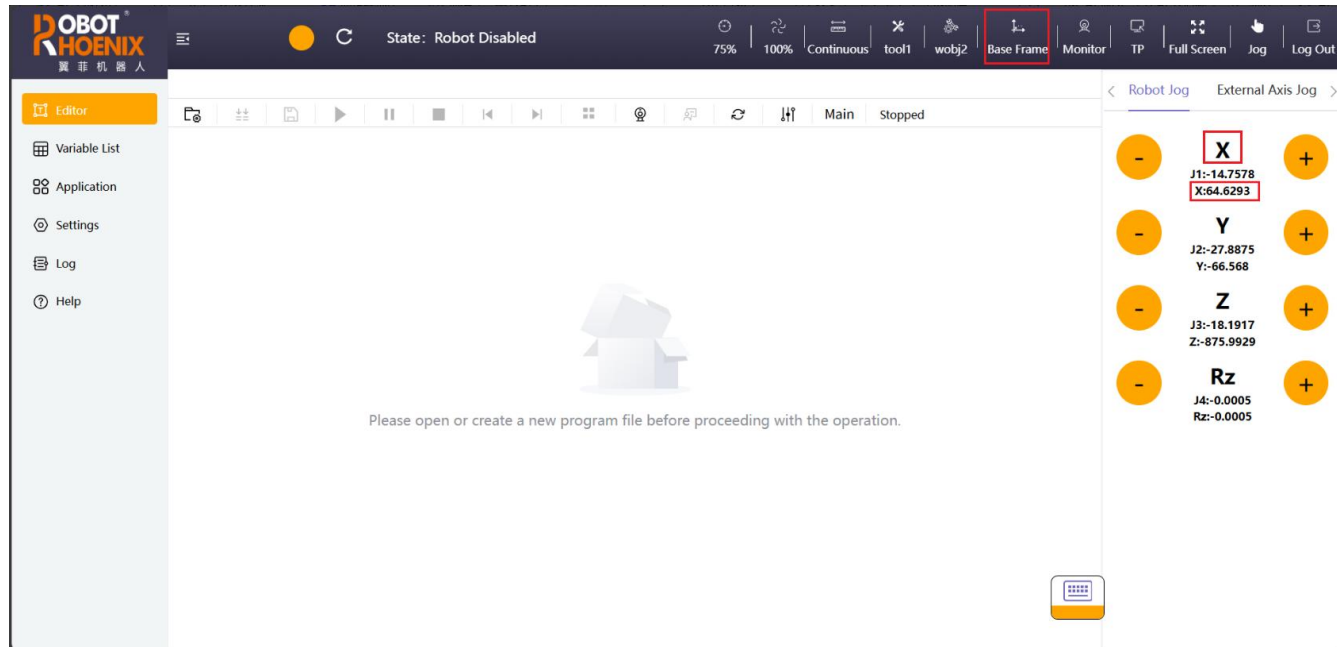
- When switched to the Joint Coordinate System, the jogging interface will display “J1”, “J2”, “J3”, “J4”, “J5”, and “J6” as the joint values of the robot. The joint values represent the rotational angles of the robot arm relative to the zero point position.

The following example shows SCARA model, which only displays “J1”, “J2”, “J3”, and “J4”.



- When switched to the Base Coordinate System, the jogging interface will display “X”, “Y”, “Z”, “RX”, “RY” and “RZ” as the position and orientation of the Tool Center Point (TCP) relative to the Base Coordinate System (X, Y, Z, RX, RY, RZ).

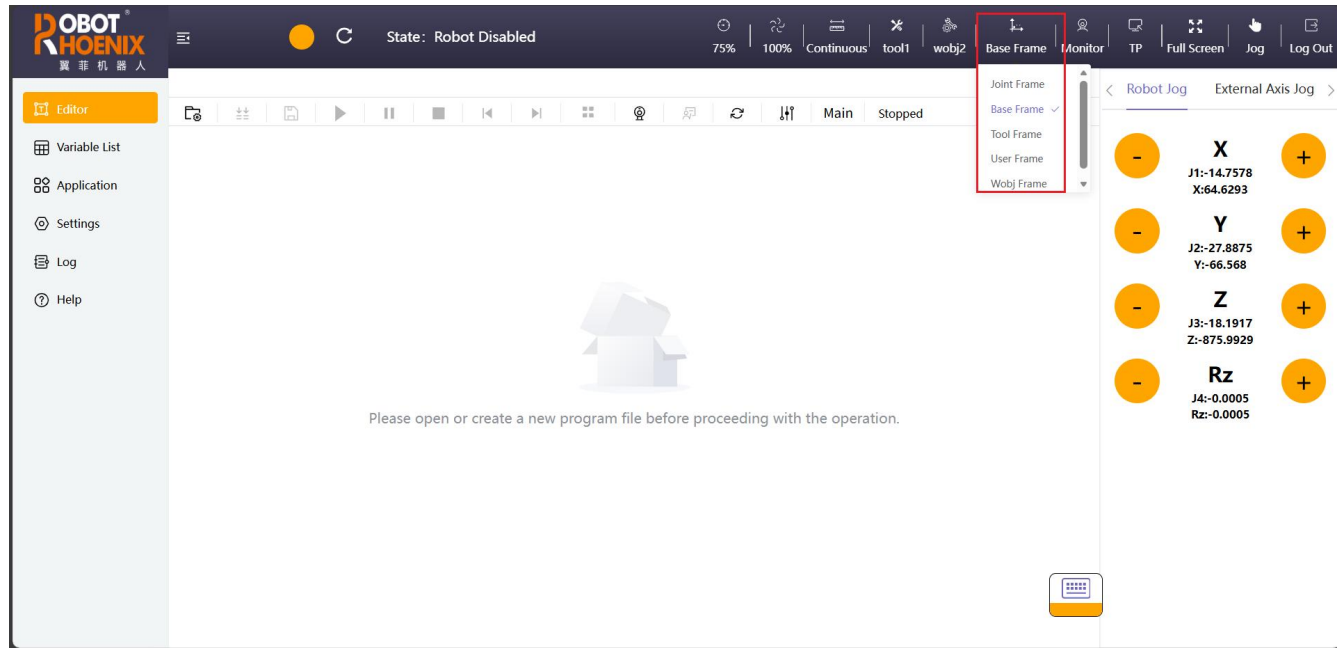
The following example shows SCARA model, which only displays “X”, “Y”, “Z” and “RZ”.



- When switched to the Tool Coordinate System, the jogging interface will display “X”, “Y”, “Z”, “RX”, “RY” and “RZ” as the position and orientation of the Tool Center Point (TCP) relative to the Base Coordinate System (X, Y, Z, RX, RY, RZ).
- When switched to the User Coordinate System, the jogging interface will display “X”, “Y”, “Z”, “RX”, “RY” and “RZ” as the position and orientation of the Tool Center Point (TCP) relative to the User Coordinate System (X, Y, Z, RX, RY, RZ).
- When switched to the Work Object Coordinate System, the jogging interface will display “X”, “Y”, “Z”, “RX”, “RY” and “RZ” as the position and orientation of the Tool Center Point (TCP) relative to the Work Object Coordinate System (X, Y, Z, RX, RY, RZ).

Operation method

As shown below, you can switch the currently used coordinate system in the top toolbar. Click the “Jog Coordinate System” drop-down box and select the target coordinate system to take effect.



Operation result

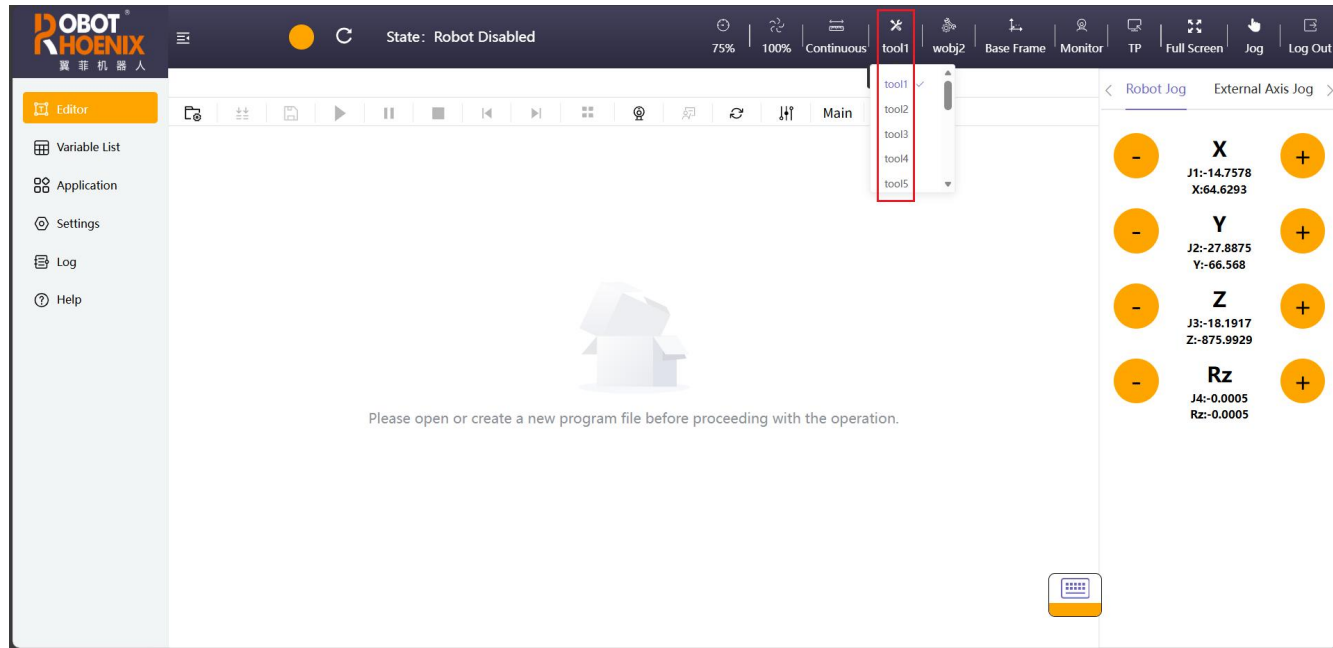
Switch to the target coordinate system. At this time, the coordinate values displayed in the jog interface are consistent with the currently selected coordinate system. Click the various control buttons in the jog interface to control the robot to move in the selected coordinate system.

1.7.5. Tool Number Switching

When changing tools during actual use, it is necessary to switch the tool number on the interface. Note that switching the tool number affects not only the robot jogging but also the teaching of the points.

Operation method

As shown below, you can switch the currently used tool number in the top toolbar, and click the “Tool Coordinate System” drop-down box to select the target tool number to take effect.

**Note:**

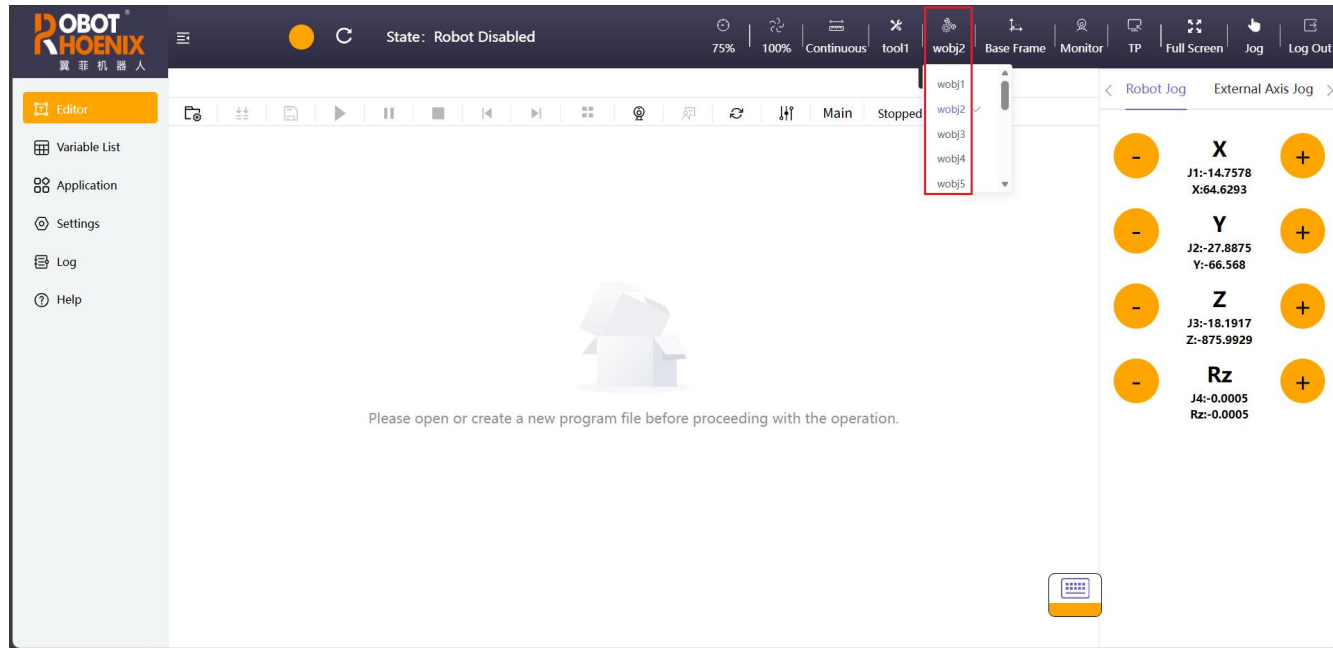
The tool number can be switched only when the control device is a “teach pendant” and no program is running.

1.7.6. Work Object Switching

In actual use, there may be multiple work object coordinate systems. Switch the work object coordinate system to match the actual application scenario. Note that the switching of the work object number is not only effective for the robot jog, but also affects the teaching of the points.

Operation method

As shown below, you can switch the currently used work object number in the top toolbar. Click the “Work object coordinate system” drop-down box to select the target work object number to take effect.

**Note:**

The work object number can be switched only when the control device is a “teach pendant” and the program is not running.

Operation result

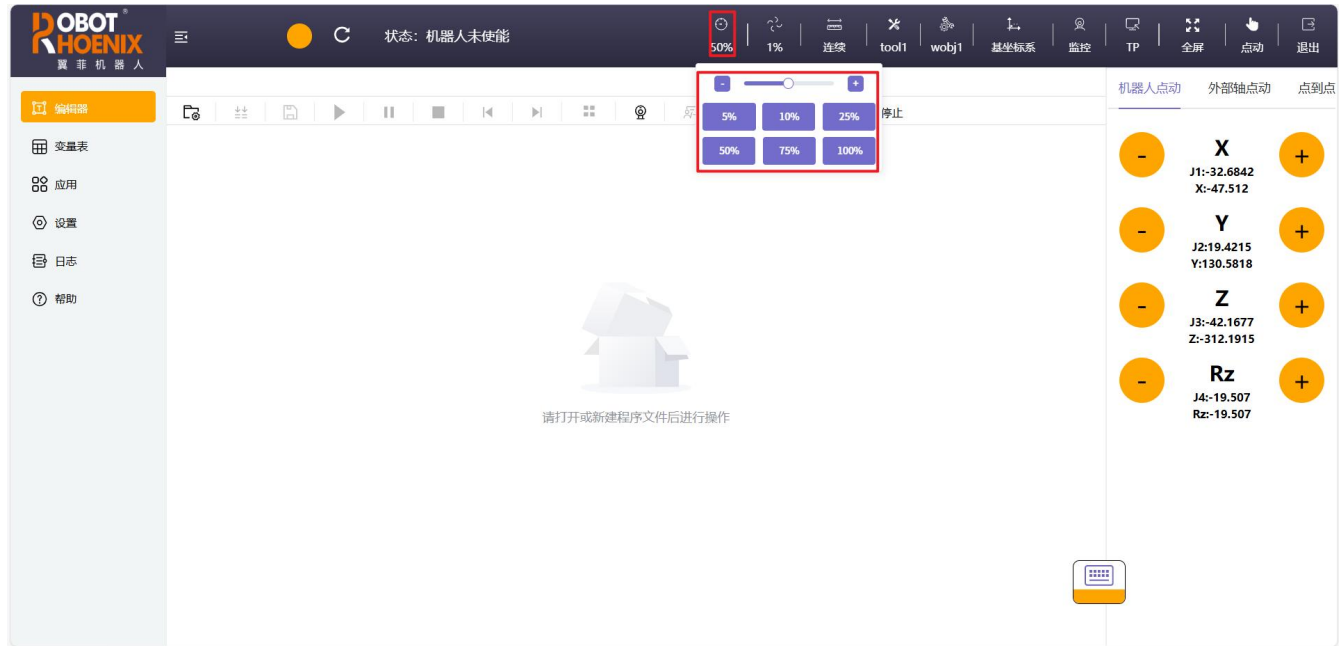
After the work object is switched, a motion test can be performed to verify whether the motion trajectory of the robot end meets the expected work object coordinate system setting.

1.7.7. Speed Adjustment

Adjust the robot’s motion speed according to the usage requirements, including the robot’s jog speed and the program’s automatic running speed.

1.7.7.1. Jog Speed Adjustment**Operation method**

As shown below, you can adjust the current robot’s jog speed in the top toolbar. Drag the slider or click “+” / “-” to set the target speed, which will take effect immediately. You can also select a preset shortcut speed.



Operation result

The speed switch takes effect immediately after completion. At this time, you can perform a jog test to verify whether the robot's motion speed meets expectations.

Note:

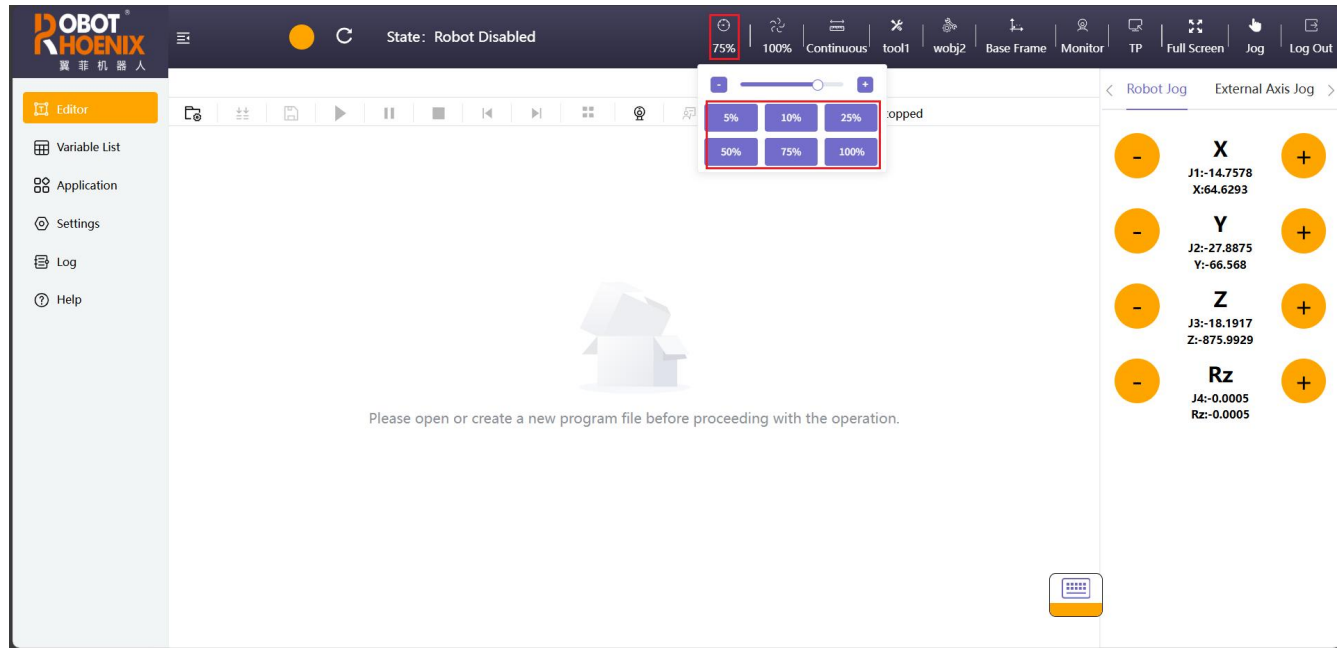
The value set for the jog speed adjustment in the joint space is the proportional value relative to the axis jog constraint. You can view the relevant parameters through "Settings" - "Robot Settings" - "Parameter Configuration" - "Axis Jog Constraint".

The value set for the jog speed adjustment in the Cartesian space is the proportional value relative to the robot jog constraint. You can view the relevant parameters through "Settings" - "Robot Settings" - "Parameter Configuration" - "Robot Jog Constraint".

1.7.7.2. Automatic Operation Speed Adjustment

Operation method

As shown below, you can adjust the current automatic running speed in the top toolbar. You can select "Low Speed" or "High Speed" mode, drag the slider, or click "+" / "-" to set the target speed. Preset shortcut speeds can also be selected.



Operation result

After switching the automatic running speed, the robot will complete the current motion command (assumed to be the first line) and the next motion command (second line) at the original speed. The new speed will take effect starting from the third motion command, at which point you can verify whether the robot's motion meets expectations.

Note:

The automatic running speed adjustment value in joint space is a proportion relative to each axis's motion constraints. You can view the related parameters via "Settings" → "Robot Settings" → "Parameter Configuration" → "Axis" → "Axis Motion Constraints".

The automatic running speed adjustment value in Cartesian space is a proportion relative to the robot's overall motion constraints. You can view the related parameters via "Settings" → "Robot Settings" → "Parameter Configuration" → "Robot Motion Constraints".

In automatic running, the High Speed mode is calculated as:

Robot Motion Constraint × Automatic Running Speed

The Low Speed mode is calculated as:

Robot Motion Constraint × Automatic Running Speed × 0.1

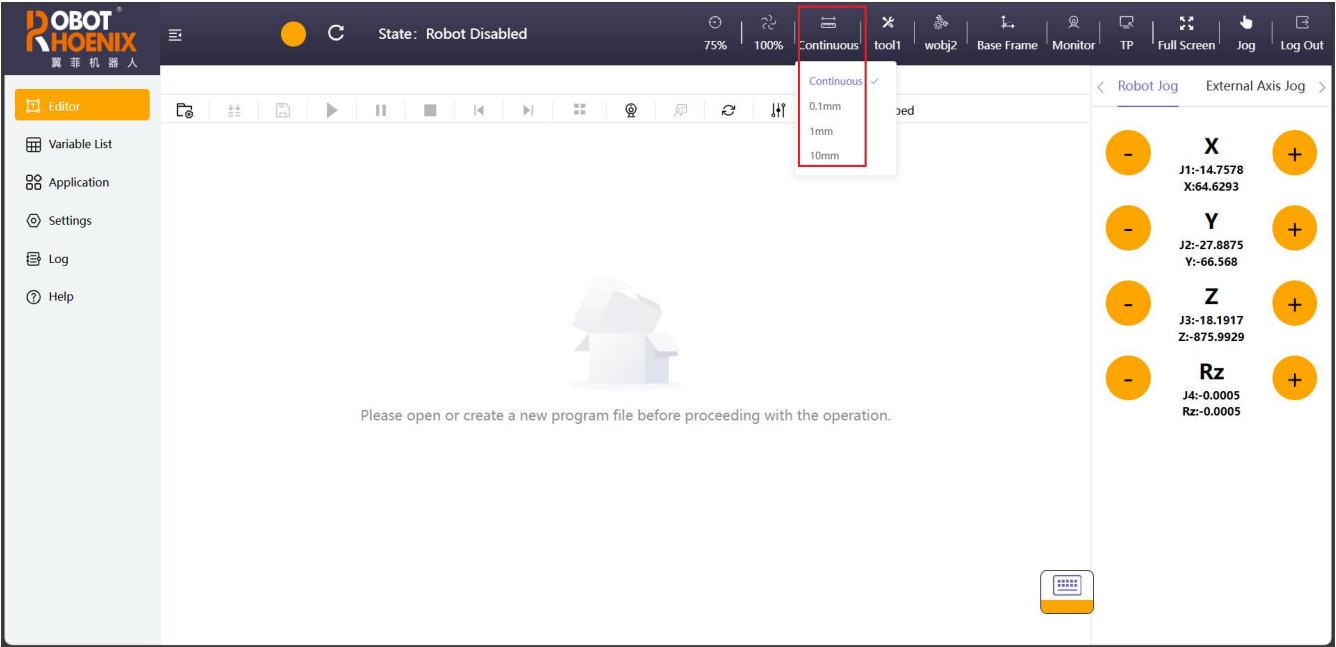
1.7.8. Jog

1.7.8.1. Robot Jog

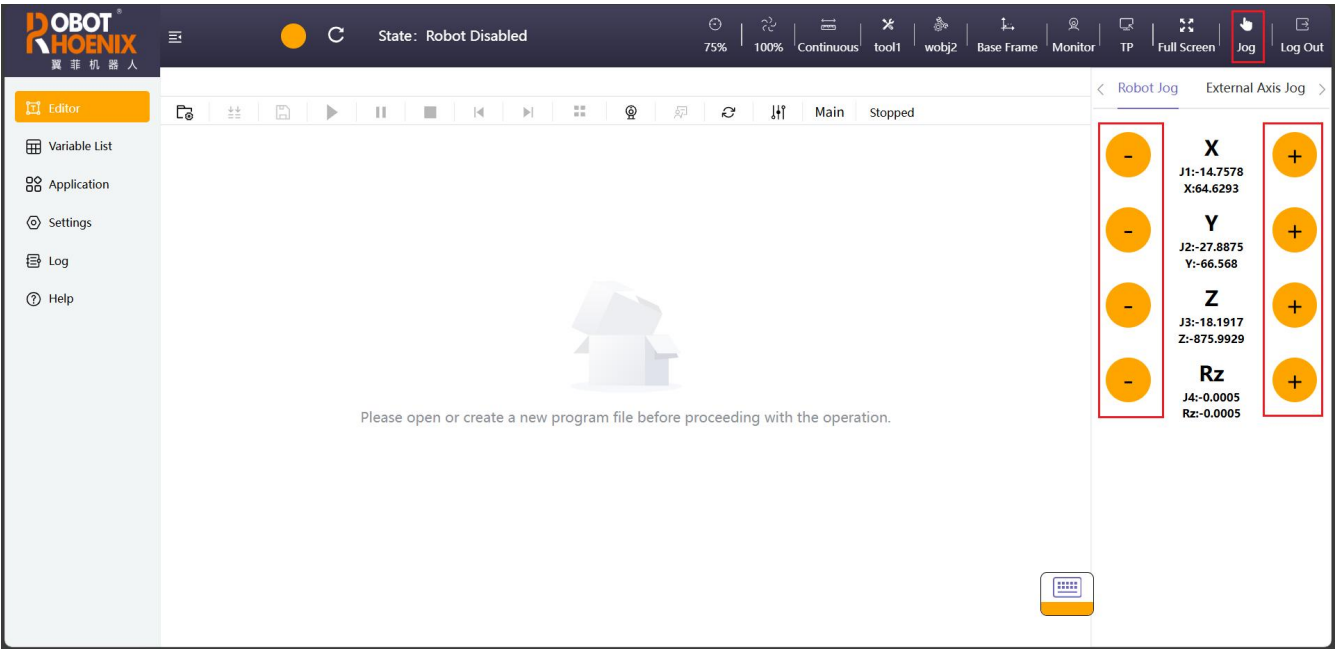
Robot jog is used to control the robot to perform continuous or inch motion.

Operation method

Step 1: As shown in the figure below, you can select the jog mode as continuous or inch motion. Both modes require setting the jog speed (14%). When inch mode is selected, you also need to set the inch distance (0.1 mm, 1 mm, or 10 mm).



Step 2: After setting the coordinate system, tool number, work object number, jog mode and jog speed, click the “Jog” button in the top toolbar to open the jog interface, click the “Robot Jog” button under the jog interface, and in the “Robot Enabled” state, click the corresponding button to start the motion (+ represents forward motion, - represents reverse motion). Press to move, and release to stop.



Note:

- Distinguish between continuous motion and inching.
- Continuous Motion: The robot moves continuously while the button is pressed and decelerates to stop when released.
- Inching means that the button is kept pressed and moves for a specified distance (the set inching distance). After moving the specified distance, it will stop regardless of whether the button is released or not, and the motion will stop if the button is released in the middle.

Inching distance

The inching distance is divided into three lengths, as shown in the following table:

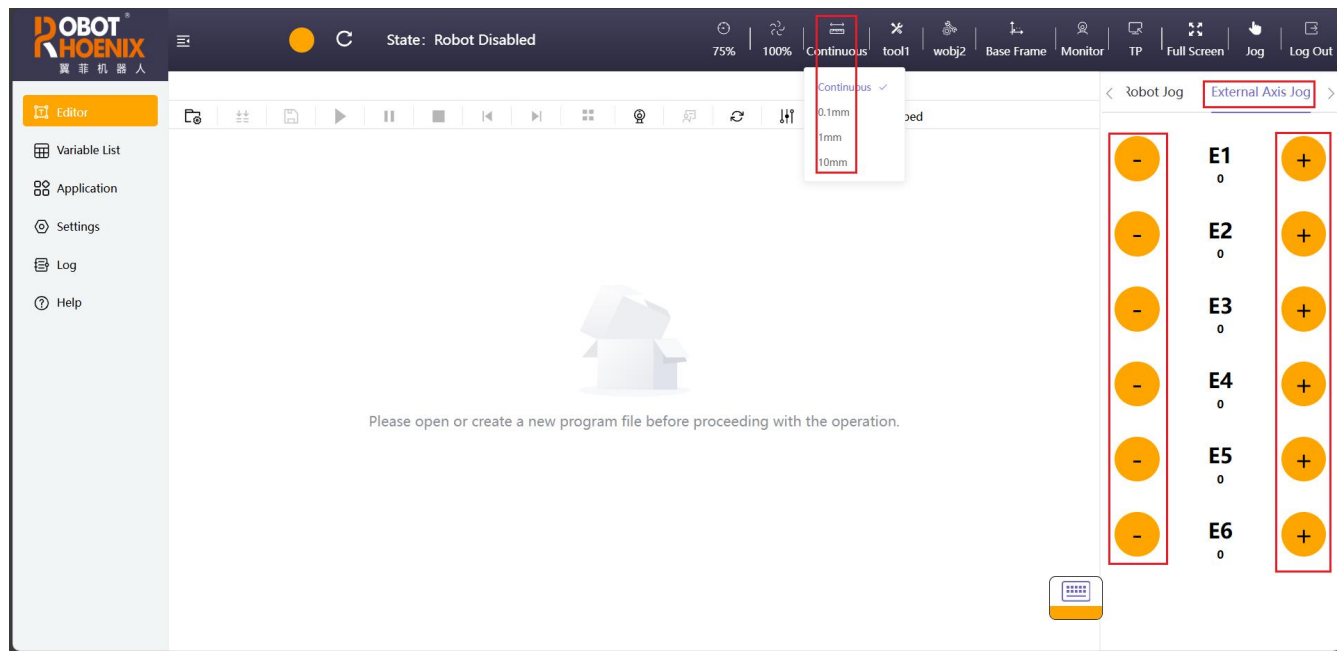
Step length	Joint Coordinate System	Cartesian Coordinate System – Linear	Cartesian Coordinate System – Rotational
10mm/10°	10°	10mm	10°
1mm/1°	1°	1mm	1°
0.1mm/0.1°	0.1°	0.1mm	0.1°

1.7.8.2. External Axis Jog

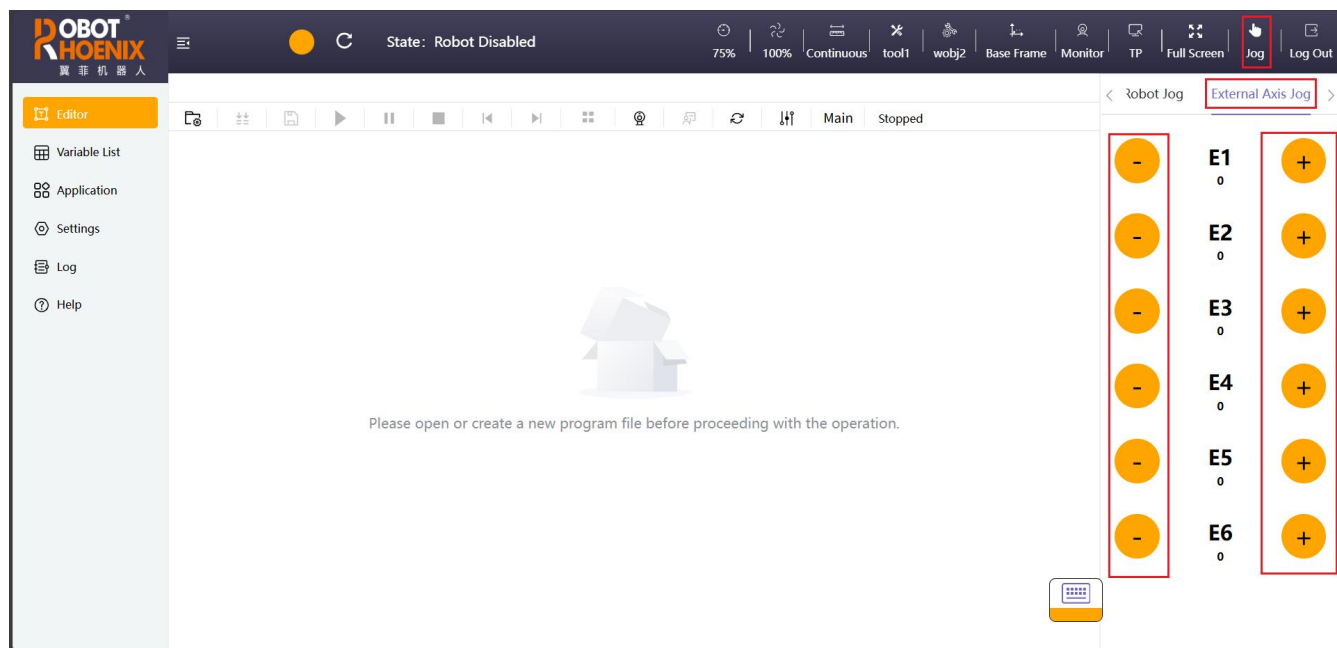
External axis jog is used to control the external axis to perform continuous motion or inching.

Operation method

Step 1: As shown below, you can select the jog mode in the top toolbar as continuous or inch motion. Both modes require setting the jog speed (). When inch mode is selected, you also need to set the inching distance (0.1 mm, 1 mm, or 10 mm).



Step 2: After setting the jog mode, inching distance and jog speed, click the “Jog” button in the top toolbar to open the jog interface. Click the “External Axis Jog” button in the jog interface. In the “Robot Enabled” state, click the corresponding external axis button to start the motion (+ represents forward motion, - represents reverse motion). Press to move, release to stop.



Note:

- Distinguish between continuous motion and inching.
- Continuous motion means that the button keeps moving as long as it is pressed, and stops immediately when it is released;
- Inching means that the button is kept pressed and moves for a specified distance (the set

inching distance). After moving the specified distance, it will stop regardless of whether the button is released or not, and the motion will stop if the button is released in the middle.

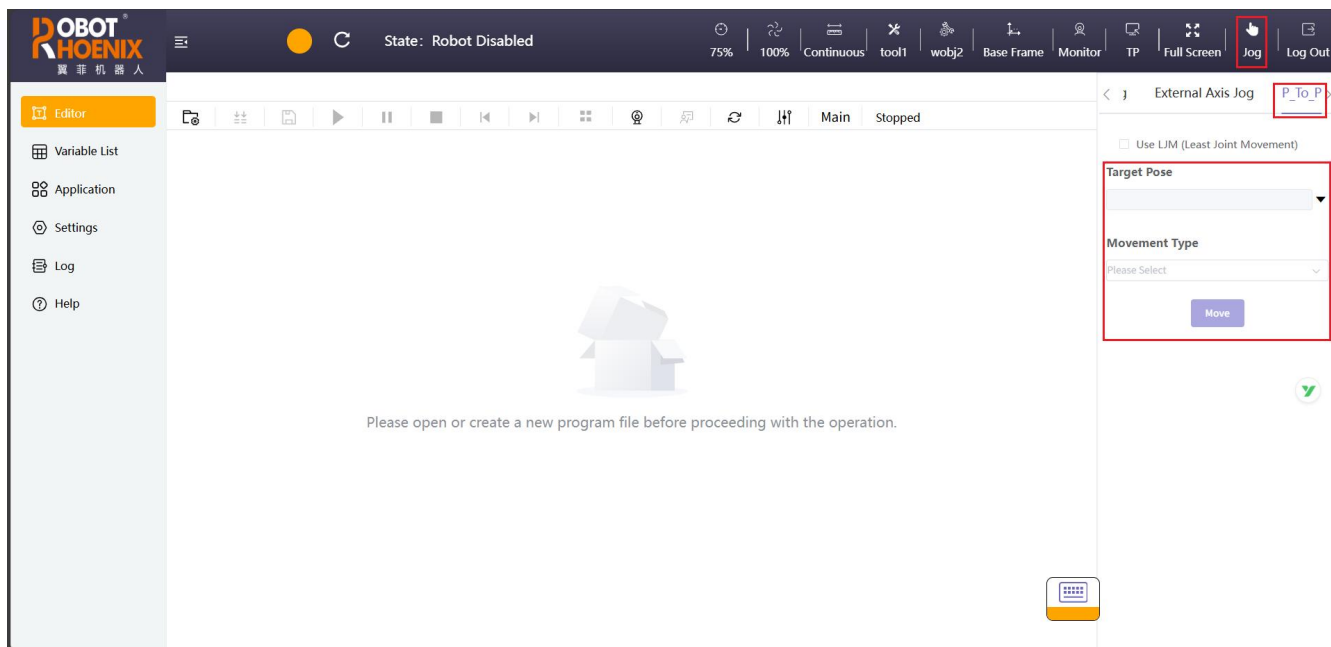
1.7.8.3. Point to Point

Point to point is used to quickly move to a designated point.

Operation method

Step 1: Add a point variable of type “RobotTarget” or “JointTarget” in the global variable table, or declare a point variable of type “RobotTarget” or “JointTarget” in the program, and keep that program file as the currently open file.

Step 2: As shown below, click the “Jog” button in the top toolbar to open the jog interface.



Step 3: In the jog interface, click the “Point-to-Point” button, expand the “Target Pose” dropdown, and select the created target point variable (global or local). Expand the “Motion Type” dropdown and choose the motion type, and select whether to enable the Minimum Joint Movement function. With the robot in the Enabled state, click the “Move” button to start motion. The robot will move continuously until it reaches the target pose and will decelerate to stop when the button is released.

Note:

- If the target pose variable type is RobotTarget, MoveL, MoveJ, and MoveLJump motion types are supported; if the target pose variable type is JointTarget, only MoveAbsJ motion type is supported.
- Pressing the “Move” button will move the robot continuously until it reaches the target

pose, and it will decelerate to stop when released.

- The Minimum Joint Movement function is only effective in non-U-axis mode.
- In U-axis mode, the robot ensures that the pose and joint positions match those during teaching when reaching the target point. Switching here does not affect the enabling status of the Minimum Joint Movement function during automatic running. By default, automatic running in non-U-axis mode has Minimum Joint Movement enabled (it is invalid in U-axis mode). To disable it, relevant commands must be used.

2. Settings

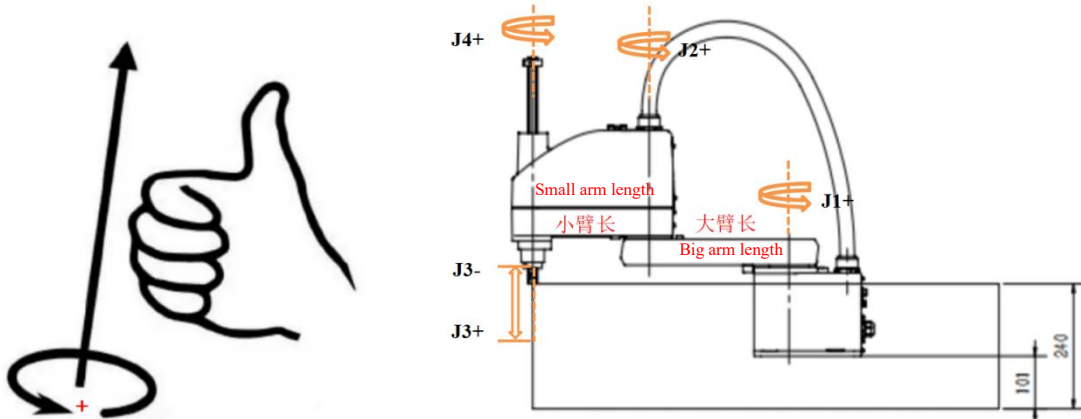
2.1. Supported Robot Models

The main robot models supported by Gorilla Studio system are: SCARA, DELTA, 6-AXIS ROBOT, DIAMOND.

2.1.1. SCARA

For SCARA robots, the rotation directions of Axis 1, Axis 2, and Axis 4 follow the right-hand rule, as shown below: point your thumb along the Z+ direction, and the direction in which your fingers curl is the positive (+) direction for Axes 1, 2, and 4. The rotation direction of Axis 3 follows the motion direction of the lead screw: downward motion is positive, and upward motion is negative.

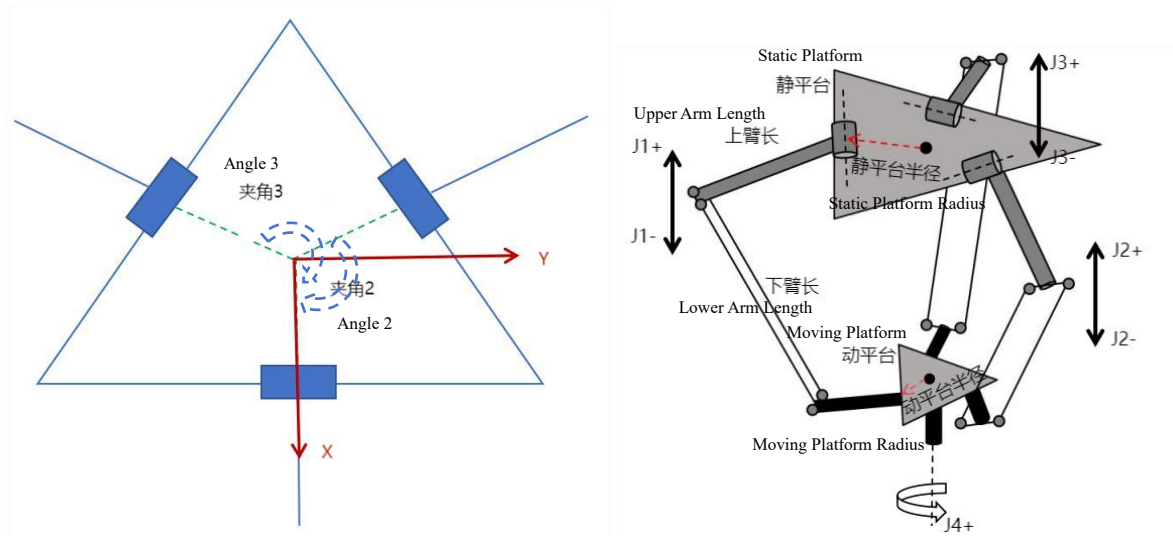
The mechanical parameters of SCARA robots include big arm length (mm), small arm length (mm), and axis 3–4 coupling ratio ($^{\circ}/\text{mm}$). The coupling ratio is defined as the number of degrees the Axis 4 joint rotates for every 1 mm movement of the Axis 3 joint.



2.1.2. DELTA

Delta robot is characterized by its compact structure, high flexibility and precision. The robot parameters include axis group speed planning method, angle 1/2/3 (unit $^{\circ}$), static platform radius (unit mm), moving platform radius (unit mm), upper arm length (unit mm), lower arm length (unit mm). J1, J2, and J3 rotate upward for positive direction, and J4 rotates

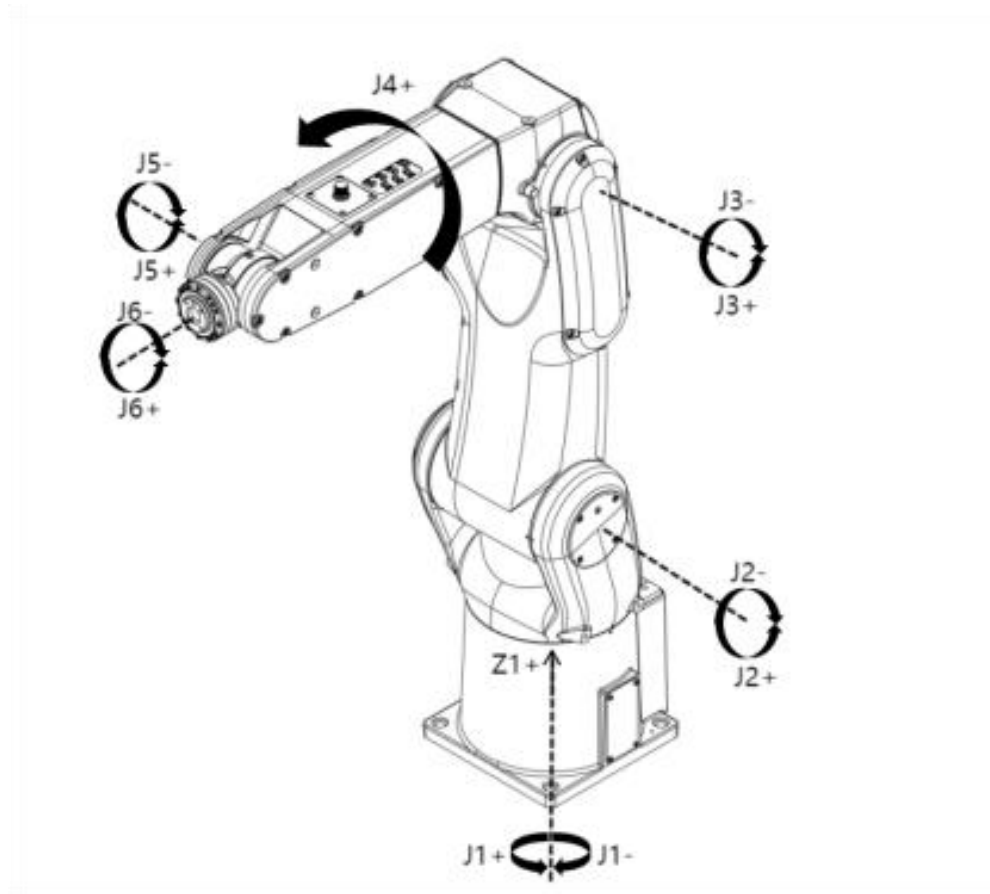
counterclockwise for positive direction.



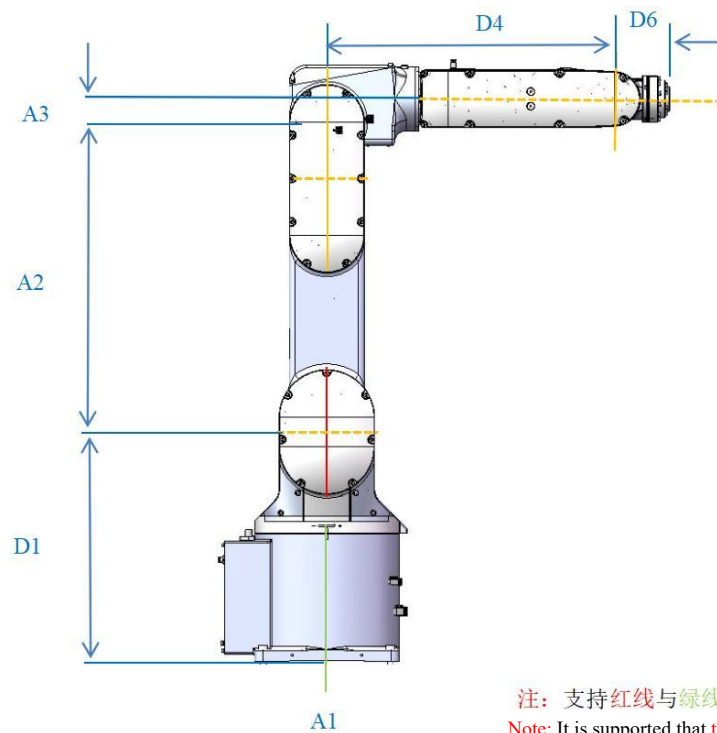
2.1.3. 6-Axis Robot

6-axis robot has six rotating joints, each driven by a servo motor, which provides a high degree of freedom, enabling it to perform complex operations and tasks. This type of robot is often used in manufacturing, automated assembly, welding, cutting, handling and other occasions that require precise control and repetitive operations.

The positive direction of each axis is shown in the figure below:



The robot parameters include DH parameters.



注：支持红线与绿线不重合。

Note: It is supported that the red line and the green line do not overlap.

The DH parameter tables for the different models of our company are shown below:

Mantis570:

Connecting rod	a	α	d	θ
1	0	-90	316	θ_1
2	270	0	0	θ_2
3	70	-90	0	θ_3
4	0	90	292	θ_4
5	0	90	0	θ_5
6	0	0	-97	θ_6

Mantis830:

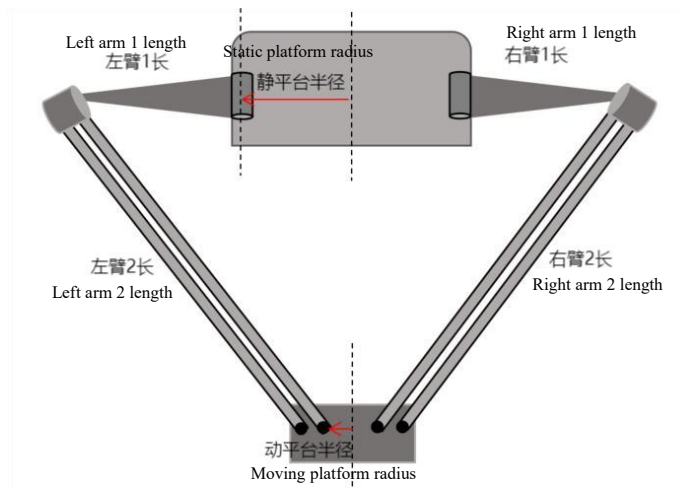
Connecting rod	a	α	d	θ
1	0	-90	411	θ_1
2	430	0	0	θ_2
3	40	-90	0	θ_3
4	0	90	395	θ_4
5	0	90	0	θ_5
6	0	0	-98	θ_6

Mantis1000:

Connecting rod	a	α	d	θ
1	0	-90	420.5	θ_1
2	510	0	0	θ_2
3	40	-90	0	θ_3
4	0	90	495	θ_4
5	0	90	0	θ_5
6	0	0	-98	θ_6

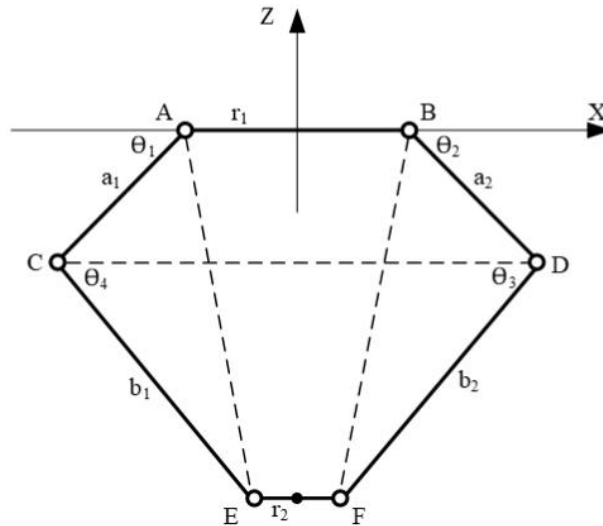
2.1.4. DIAMOND

The DIAMOND mechanism is similar to the DELTA mechanism, except that the DIAMOND mechanism is a two-arm parallel mechanism that contains two rotation axes and can realize the positioning and movement of the robot on a plane. Due to its simple structure, this type of robot is often used in industrial automation, precision operation, and teaching. Compared with other types of robots, such as serial robots, delta robots have better rigidity and faster motion speed.



As shown below, specifically, when $r_1=r_2=0$, point A coincides with point B, point E coincides with point F, and the mechanism is simplified to a five-bar mechanism.

Explanation of the positive and negative joint angles: θ_1 and θ_2 are negative below the X-axis and positive above the X-axis.

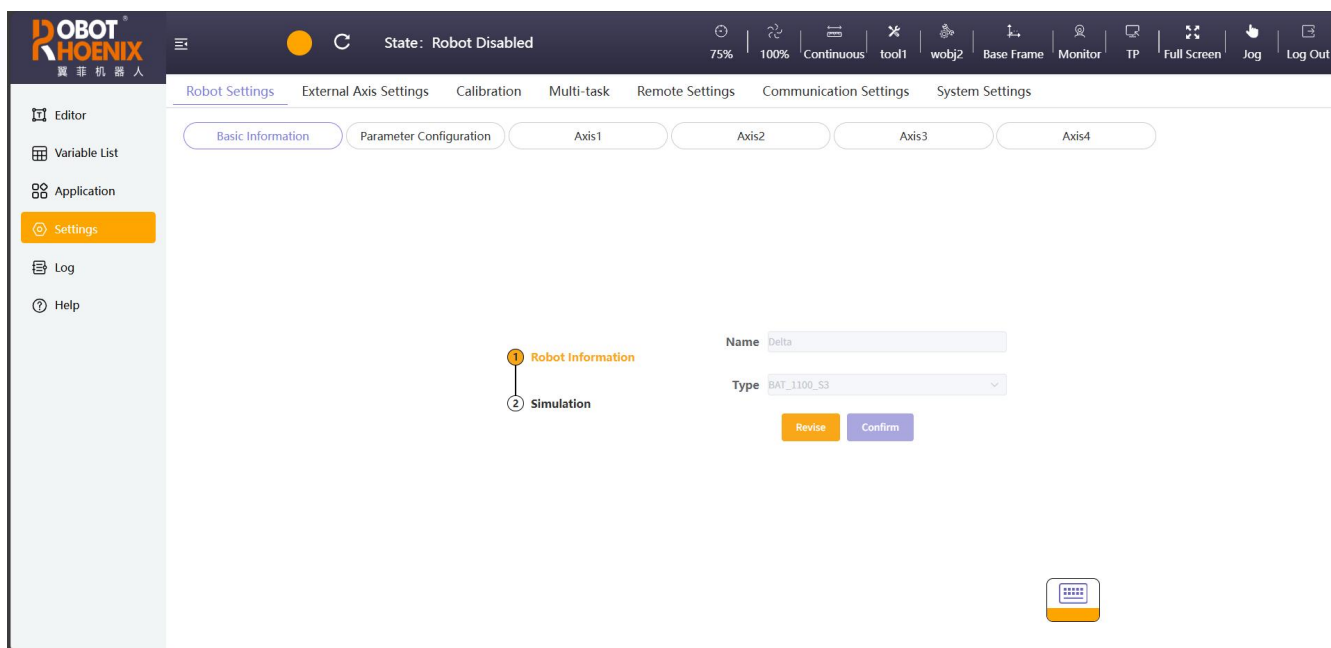


2.2. Robot Settings

When the robot status is “Robot not enabled”, the robot settings can be modified. “Robot settings” include robot basic information settings, robot parameter configuration, and axis parameter settings.

2.2.1. Basic Information

The basic information includes the robot name, robot type, and whether the simulation mode is enabled. The robot name cannot be modified.

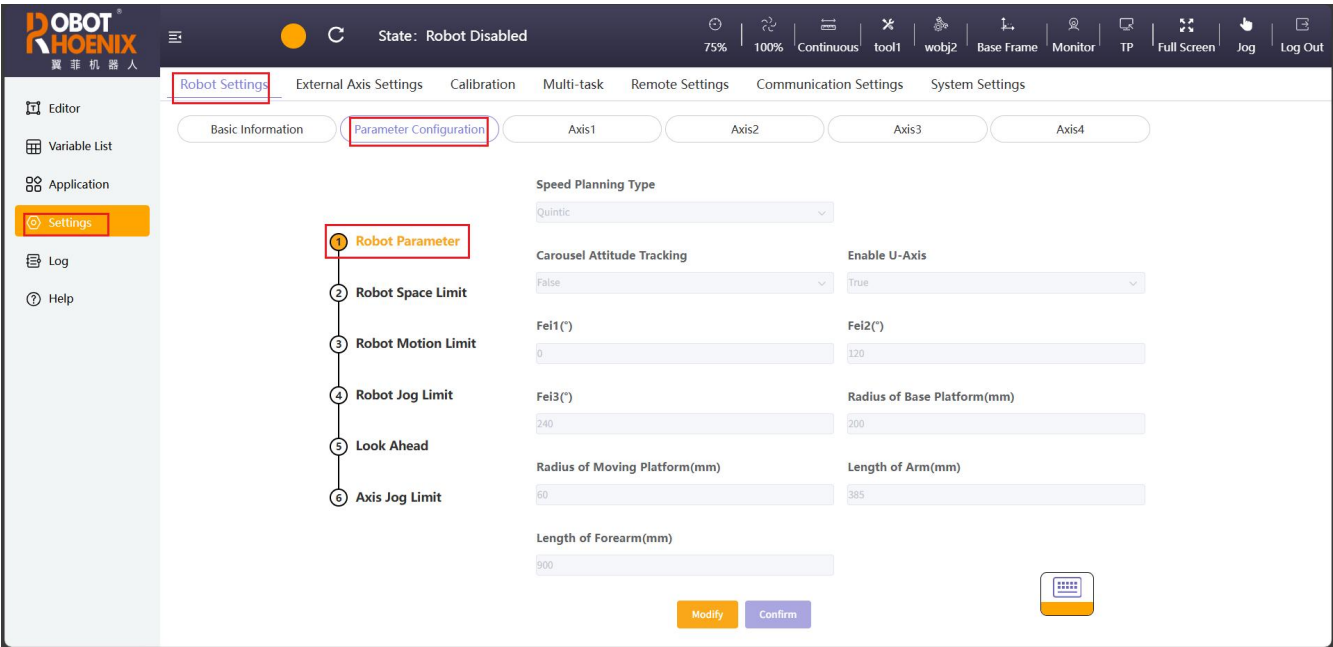


2.2.2. Parameter Configuration

2.2.2.1. Robot Parameters

Operation method

Click “Settings” - “Robot Settings” - “Parameter Configuration” - “Robot Parameters” in the left navigation bar to enter the robot parameter configuration interface, click the “Modify” button to modify, and click “OK” after the modification is completed.



The parameter configuration of each model is as follows:

- SCARA

Items	Description
Axis group speed planning method	Supports Modified_S_Curve and Quintic
Enable U-axis	True to enable, False to disable
Disc orientation following	True to enable, False to disable
Big arm length (unit: mm)	Default values have been set at the factory.
Small arm length (unit: mm)	Default values have been set at the factory.
Coupling ratio (unit: °/mm)	Default values have been set at the factory.

- DELTA

Items	Description
Axis group speed planning method	Supports Modified_S_Curve and Quintic
Enable U-axis	True to enable, False to disable
Disc orientation following	True to enable, False to disable
Angle 1 (unit: °)	Default values have been set at the factory.
Angle 2 (unit: °)	Default values have been set at the factory.
Angle 3 (unit: °)	Default values have been set at the factory.
Static platform radius (unit: mm)	Default values have been set at the factory.
Moving platform radius (unit: mm)	Default values have been set at the factory.
Upper arm length (unit: mm)	Default values have been set at the factory.
Lower arm length (unit: mm)	Default values have been set at the factory.

- 6-Axis Robot

Items	Description
Axis group speed planning method	Supports Modified_S_Curve and Quintic
Disc orientation following	True to enable, False to disable
1/2/3/4/5/6 axis Alpha	Default values have been set at the factory.
1/2/3/4/5/6 axis axis A	Default values have been set at the factory.
1/2/3/4/5/6 axis axis D	Default values have been set at the factory.

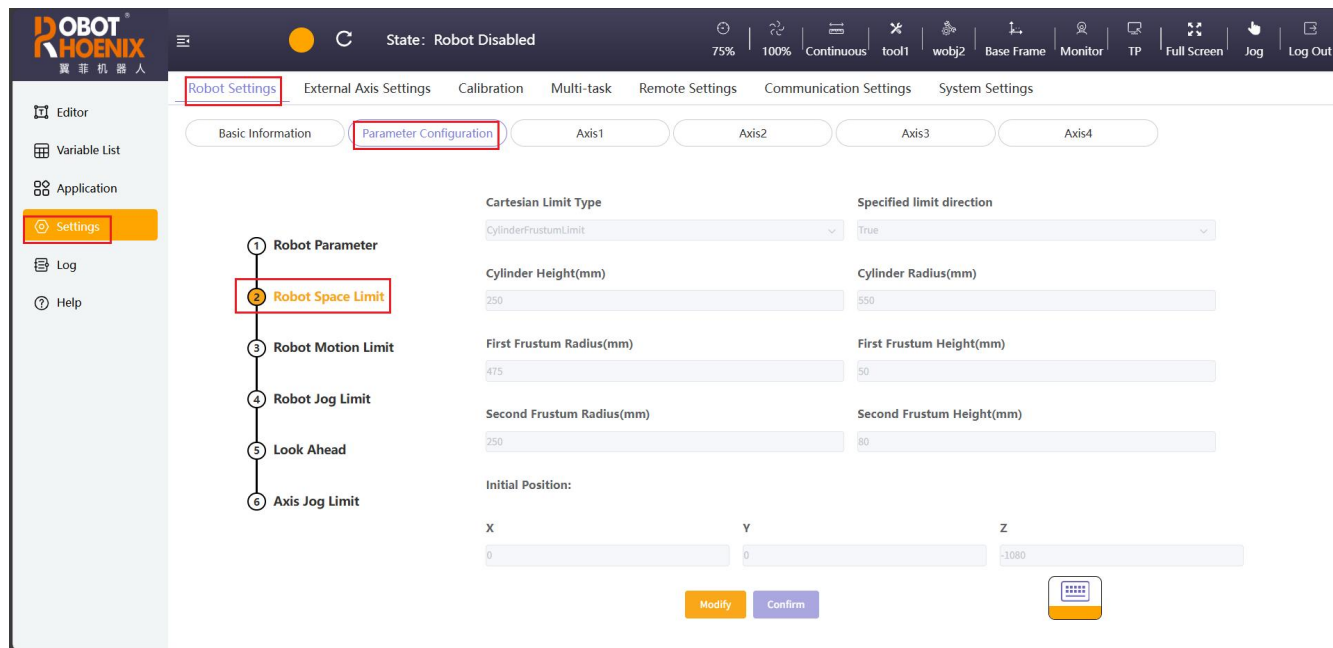
- DIAMOND

Items	Description
Axis group speed planning method	Supports Modified_S_Curve and Quintic
Enable U-axis	True to enable, False to disable
Disc orientation following	True to enable, False to disable
Left arm 1 length (unit: mm)	Default values have been set at the factory.
Left arm 2 length (unit: mm)	Default values have been set at the factory.
Right arm 1 length (unit: mm)	Default values have been set at the factory.
Right arm 2 length (unit: mm)	Default values have been set at the factory.
Static platform radius (unit: mm)	Default values have been set at the factory.
Moving platform radius (unit: mm)	Default values have been set at the factory.

2.2.2.2. Robot Space Constraints

Operation method

Click “Settings” - “Robot Settings” - “Parameter Configuration” - “Robot Space Constraints” in the left navigation bar to enter the robot space constraint settings interface, click the “Modify” button to modify, and click “OK” after the modification is completed. As shown below:



Description

Gorilla Studio system supports three types of Cartesian space constraints: NoLimit, ZDirectionLimit, and CylinderFrustumLimit.

- NoLimit

Description

Unlimited.

Note:

Setting this type may result in uncontrollable motion or serious accidents, so please operate with caution.

- ZDirectionLimit

Description

Cartesian space Z direction limit.

Parameters

Positive limit in Z direction of Cartesian space: unit: mm.

Negative limit in Z direction of Cartesian space: unit: mm.

● CylinderFrustumLimit

Description

Cartesian space frustum limit.

Parameters

Specify the limit direction: True means the cylinder is on the top and the frustum is on the bottom; False means the frustum is on the top and the cylinder is on the bottom.

Cylinder Height: The value must be greater than 0, in mm.

Cylinder Radius: The value must be greater than 0, in mm.

First Frustum Radius: The value must be greater than or equal to 0, in mm.

First Frustum Height: The value must be greater than or equal to 0, in mm.

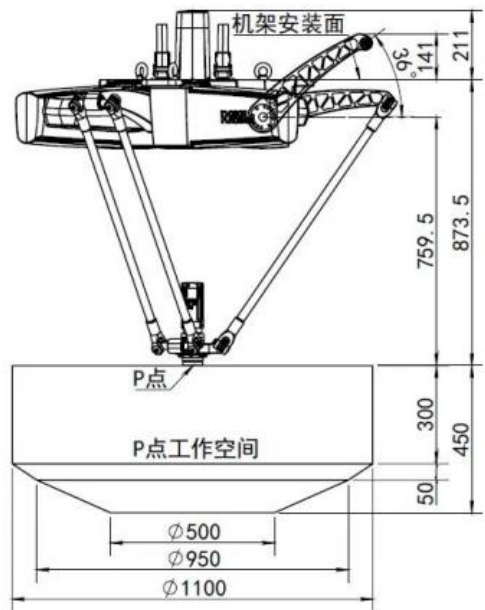
Second Frustum Radius: The value must be greater than or equal to 0, in mm.

Second Frustum Height: The value must be greater than or equal to 0, in mm.

Initial Position X, Y, Z: The X, Y, Z coordinates of the center of the small end surface of the frustum.

Example

Taking Bat1100M-S15 as an example, as shown below:



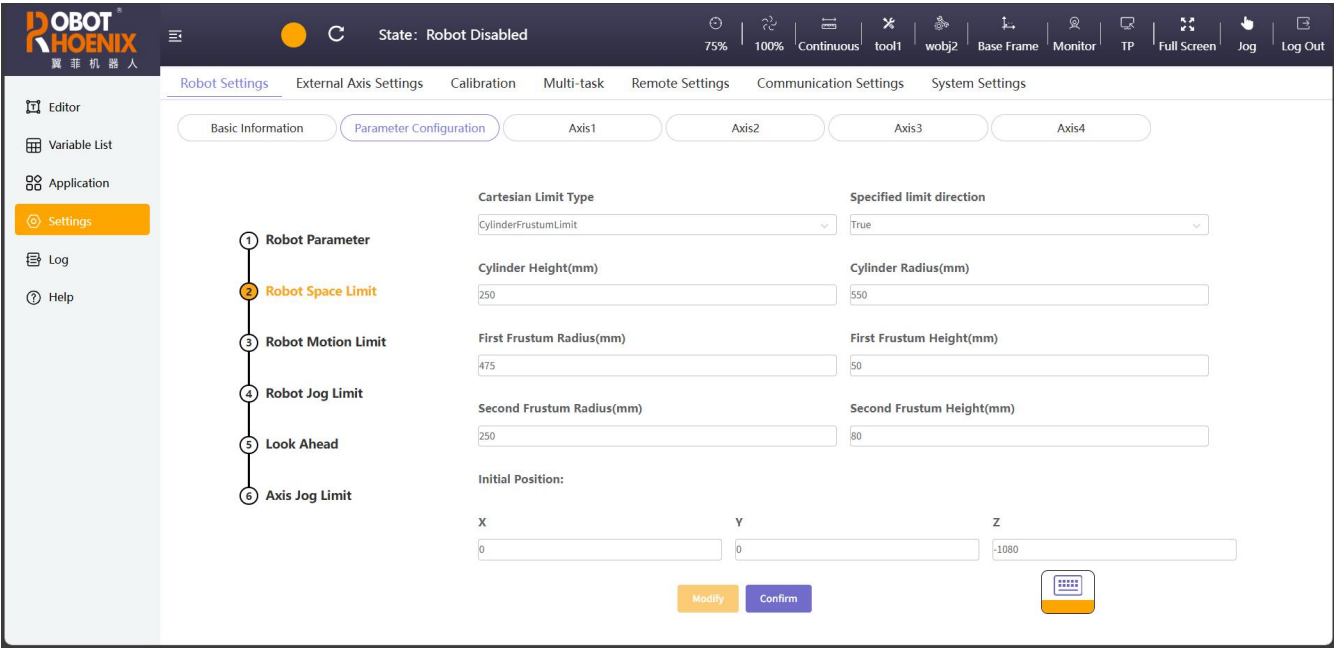
Distance between the reducer center and the moving-platform joint center: 700 mm
减速机中心与动平台关节中心距离: 700mm

型号	安装面至设备顶端距离	上臂最大抬升与水平面角度	上臂最大抬升至安装面高度	安装面至工作空间上平面距离	减速机轴心至工作空间上平面距离	工作空间上下极限总高度	工作空间圆柱形部分高度	工作空间锥形台部分高度	工作空间锥形柱形过渡直径	工作空间锥形柱形底部直径	工作空间圆形台部分直径	机器人本体最小安装尺寸
Bat1100M-S15	211mm	36°	141mm	873.5mm	759.5mm	450mm	300mm	50mm	950mm	500mm	1100mm	1400mm

Model	Distance from installation surface to top of equipment	Max. upper arm elevation and horizontal plane angle	Max. elevation of the upper arm to the height of the installation surface	Distance from installation surface to upper plane of working space	Distance from the reducer shaft center to the upper plane of the working space	Total height of the upper and lower limits of the working space	Height of the cylindrical part of the working space	Height of the conical frustum part of the working space	Transition diameter of the conical column in the working space	Bottom diameter of the conical column in the working space	Diameter of the circular table part of the working space	Min. installation size of the robot body
Bat1100M-S15	211 mm	36°	141 mm	873.5 mm	759.5 mm	450 mm	300 mm	50 mm	950 mm	500 mm	1100 mm	1400 mm

As shown in the figure, set the “Cartesian Space Limit Type” to “CylinderFrustumLimit”. For this robot model, the actual restricted space consists of a cylinder on the top and a frustum on the bottom. Set “Specify Limit Direction” to “True”, the “Cylinder Height” to 300 mm, and the “Cylinder Radius” to 550 mm (1100/2). The “First Frustum Radius” should be set to 475 mm (950/2) with a “First Frustum Height” of 50 mm, and the “Second Frustum Radius” should be set to 250 mm (500/2) with a “Second Frustum Height” of 100 mm. The “Initial Position”, which is the center point of the small end of the frustum, should be set to (0, 0, -1150). Once

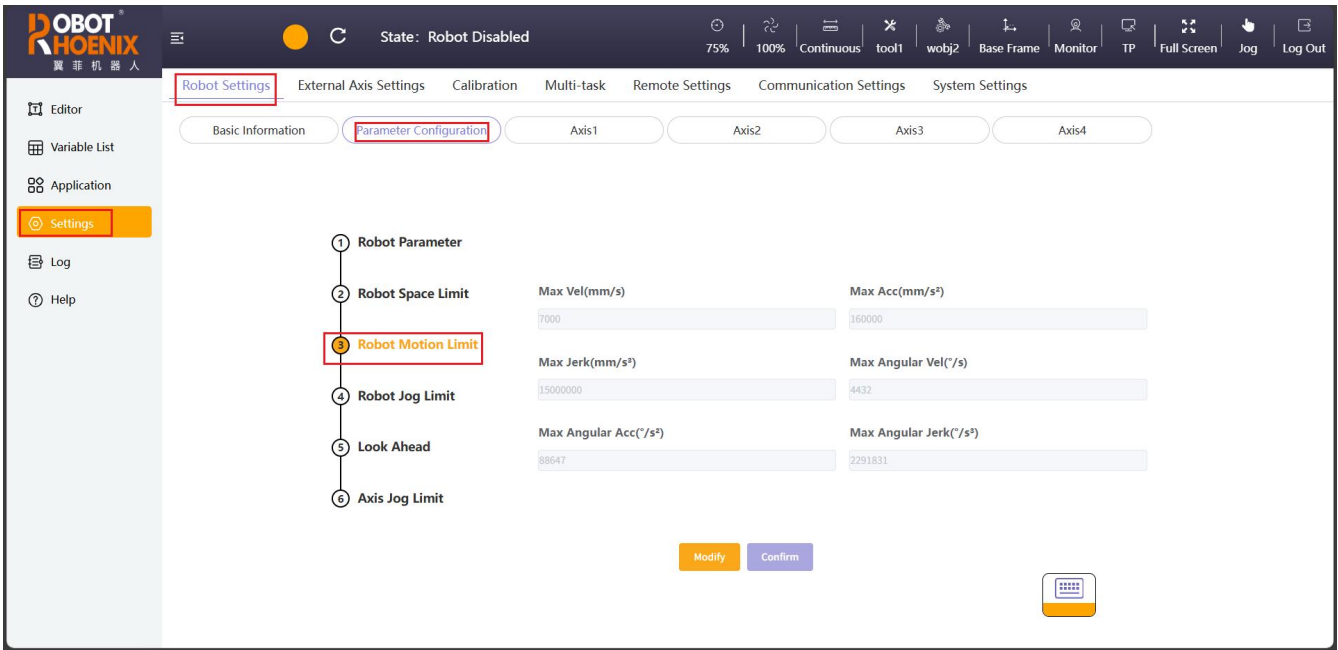
the settings take effect, the system will generate an error if the robot's position exceeds this restricted space.



2.2.2.3. Robot Motion Constraints

Operation method

Click "Settings" - "Robot Settings" - "Parameter Configuration" - "Robot Motion Constraints" in the left navigation bar to enter the robot motion constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. As shown below:



Description

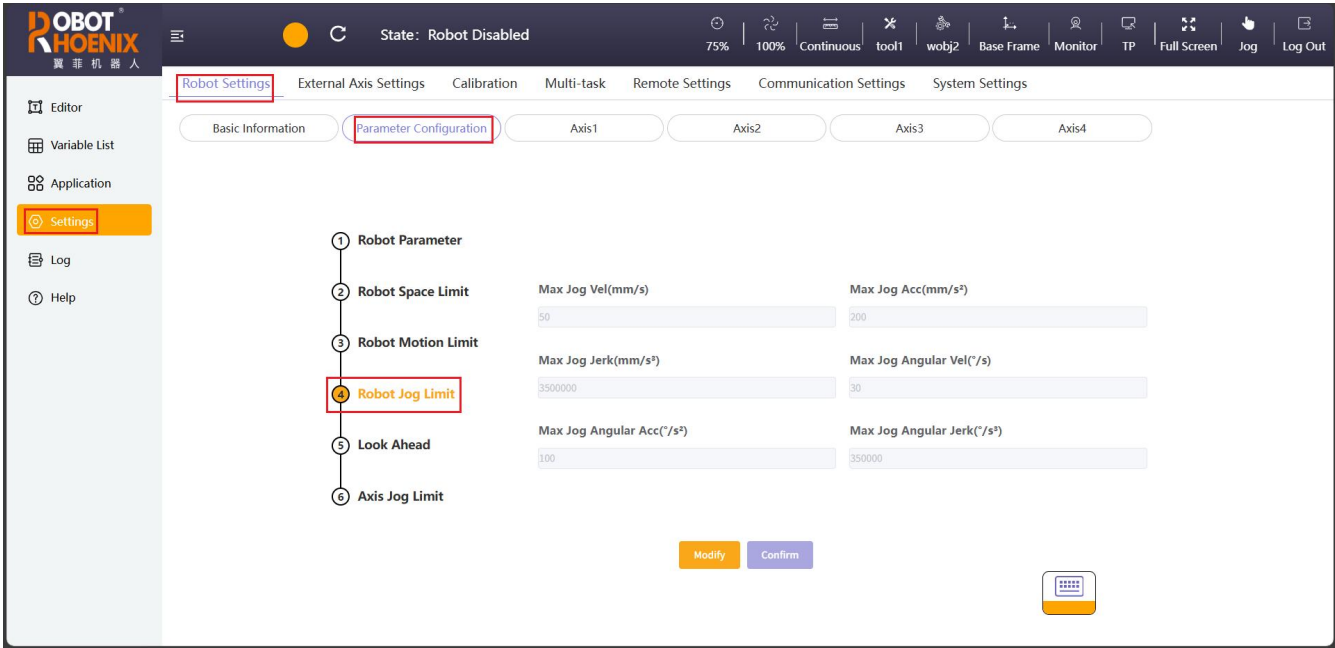
Items	Description
Maximum linear speed (unit: mm/s)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum linear acceleration (unit: mm/s ²)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum linear jerk (unit: mm/s ³)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum orientation angular velocity (unit: °/s)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum orientation angular acceleration (unit: °/s ²)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum orientation angular jerk (unit: °/s ³)	Default values have been set at the factory, and the set value must be greater than 0.

2.2.2.4. Robot Jog Constraints

Operation method

Click "Settings" - "Robot Settings" - "Parameter Configuration" - "Robot Jog Constraints" in the left navigation bar to enter the robot jog constraint settings interface, click the "Modify"

button to modify, and click "OK" after the modification is completed. As shown below:



Description

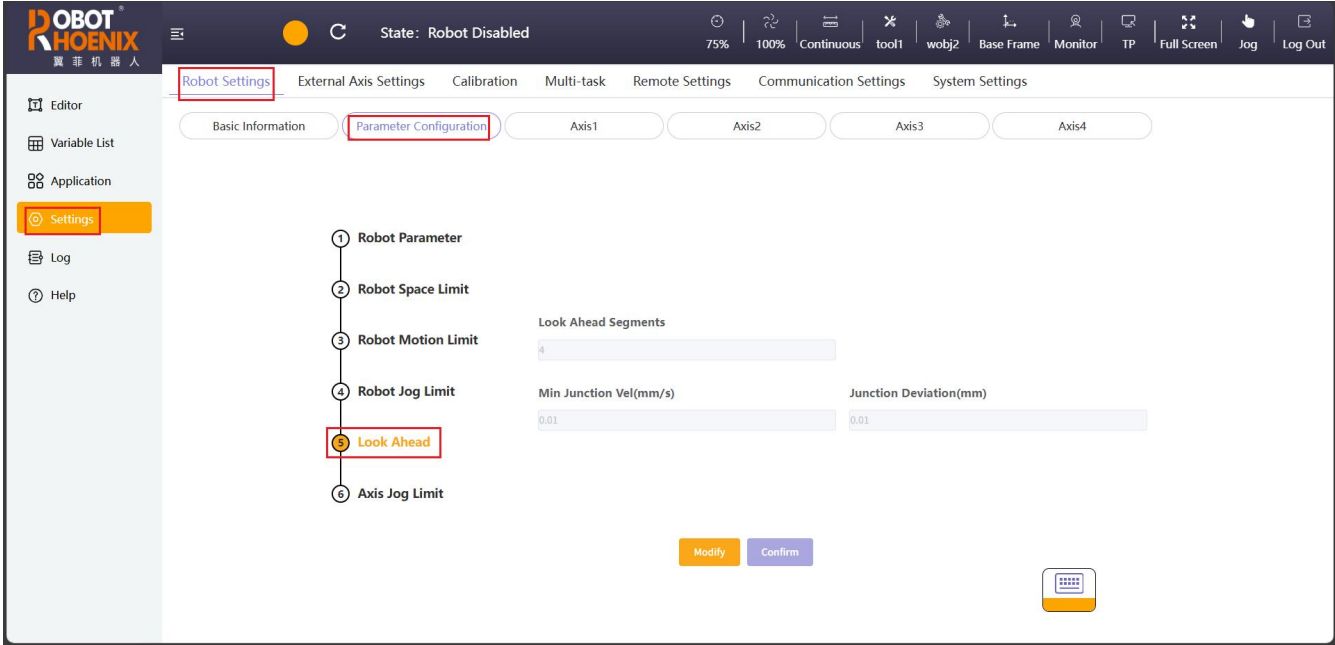
Items	Description
Maximum jog linear speed (unit: mm/s)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum jog linear acceleration (unit: mm/s ²)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum jog linear jerk (unit: mm/s ³)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum jog rotation speed (unit: °/s)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum jog rotation acceleration (unit: °/s ²)	Default values have been set at the factory, and the set value must be greater than 0.
Maximum jog rotation jerk (unit: °/s ³)	Default values have been set at the factory, and the set value must be greater than 0.

2.2.2.5. Look-ahead

Look-ahead (MovePath) means that during the robot's motion, the control system processes the program commands after the current command being executed by the robot in advance.

Operation method

Click "Settings" - "Robot Settings" - "Parameter Configuration" - "MovePath" in the left navigation bar to enter the look-ahead settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. As shown below:



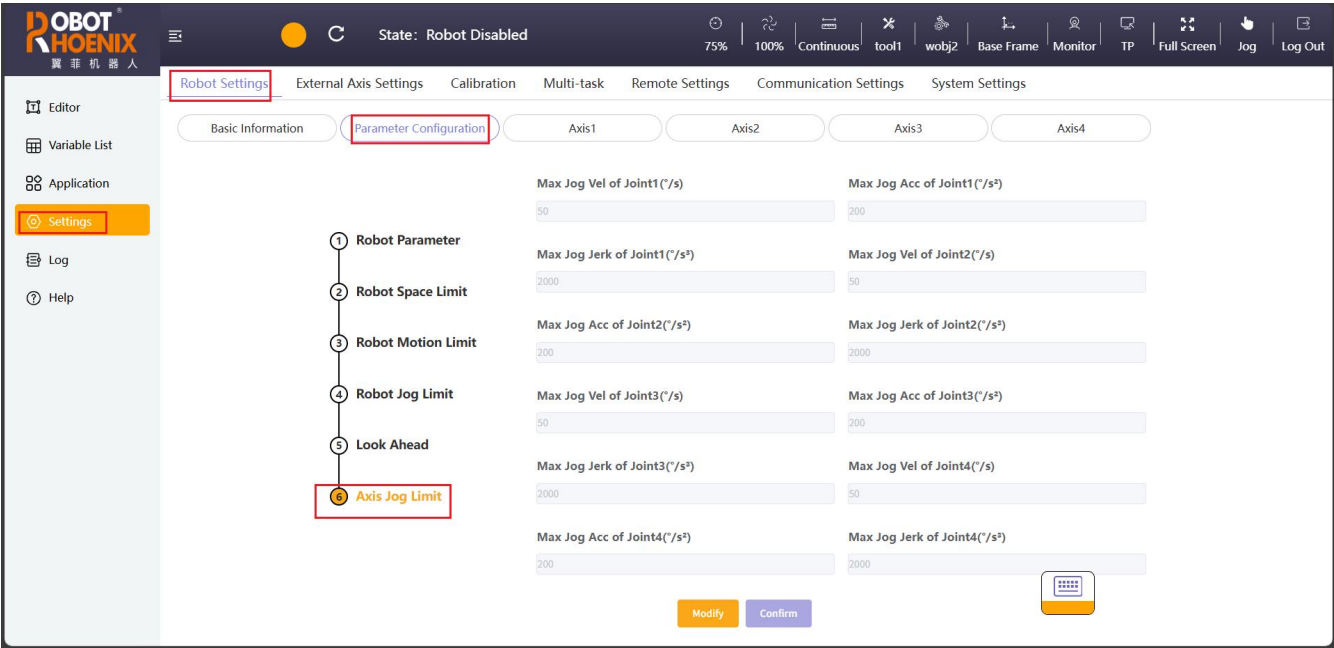
Description

Items	Description
Look-ahead commands number	Must be an even number, valid range is [2,500].
Minimum corner velocity (unit: mm/s)	The speed during processing shall not be lower than this value, and the valid range is (0,9].
Allowable corner error (unit: mm)	The smaller the error value is, the slower the speed and the higher the processing accuracy. The valid range is (0,1].

2.2.2.6. Axis Jog Constraints

Operation method

Click "Settings" - "Robot Settings" - "Parameter Configuration" - "Axis Jog Constraints" in the left navigation bar to enter the axis jog constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. As shown below:



Description

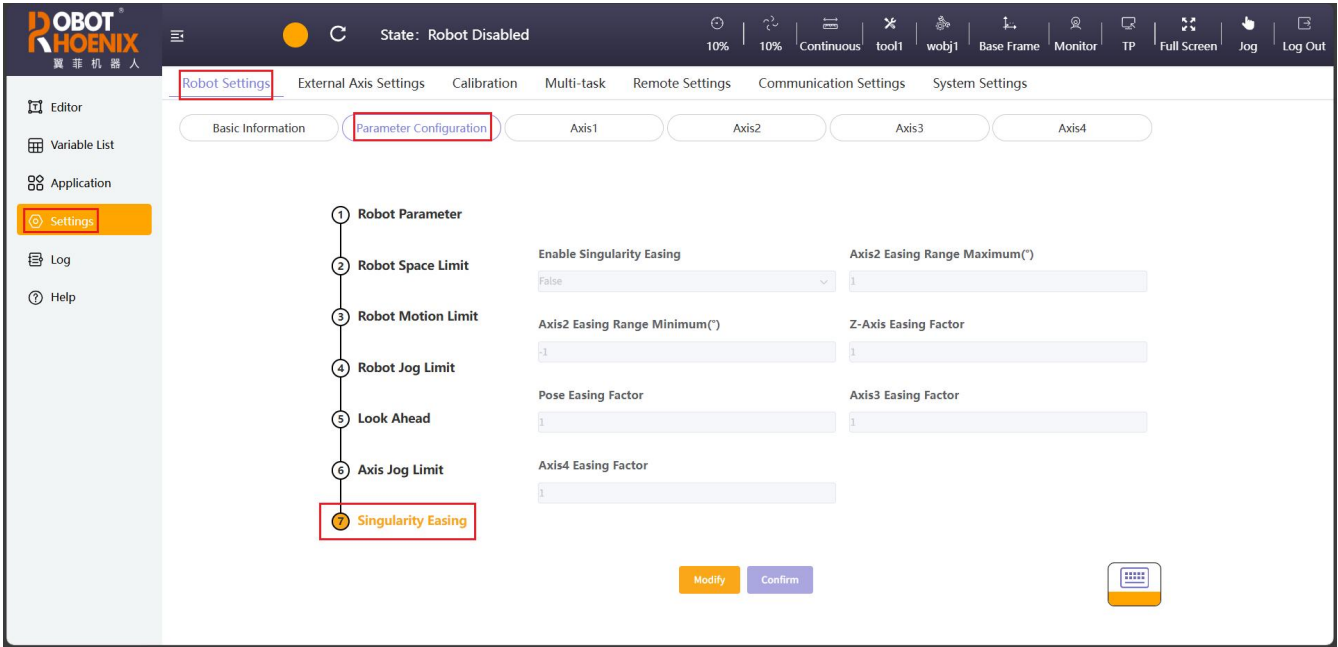
Items	Description
Maximum speed/acceleration/jerk for axis 1 jog	Default values have been set at the factory, and the set value must be greater than 0.
Maximum speed/acceleration/jerk for axis 2 jog	Default values have been set at the factory, and the set value must be greater than 0.
Maximum speed/acceleration/jerk for axis 3 jog	Default values have been set at the factory, and the set value must be greater than 0.
Maximum speed/acceleration/jerk for axis 4 jog	Default values have been set at the factory, and the set value must be greater than 0.
Maximum speed/acceleration/jerk for axis 5 jog	Default values have been set at the factory, and the set value must be greater than 0.
Maximum speed/acceleration/jerk for axis 6 jog	Default values have been set at the factory, and the set value must be greater than 0.

2.2.2.7. Singularity Easing

When the robot arm approaches a singularity, enabling this function automatically reduces the motion performance. This function is applicable only to SCARA robots.

Operation method

Click "Settings" - "Robot Settings" - "Parameter Configuration" - "Singularity Easing" in the left navigation bar to enter the configuration page. Click “Edit” to make changes, and click “Confirm” after editing to save the settings. As shown in the figure below:



Description

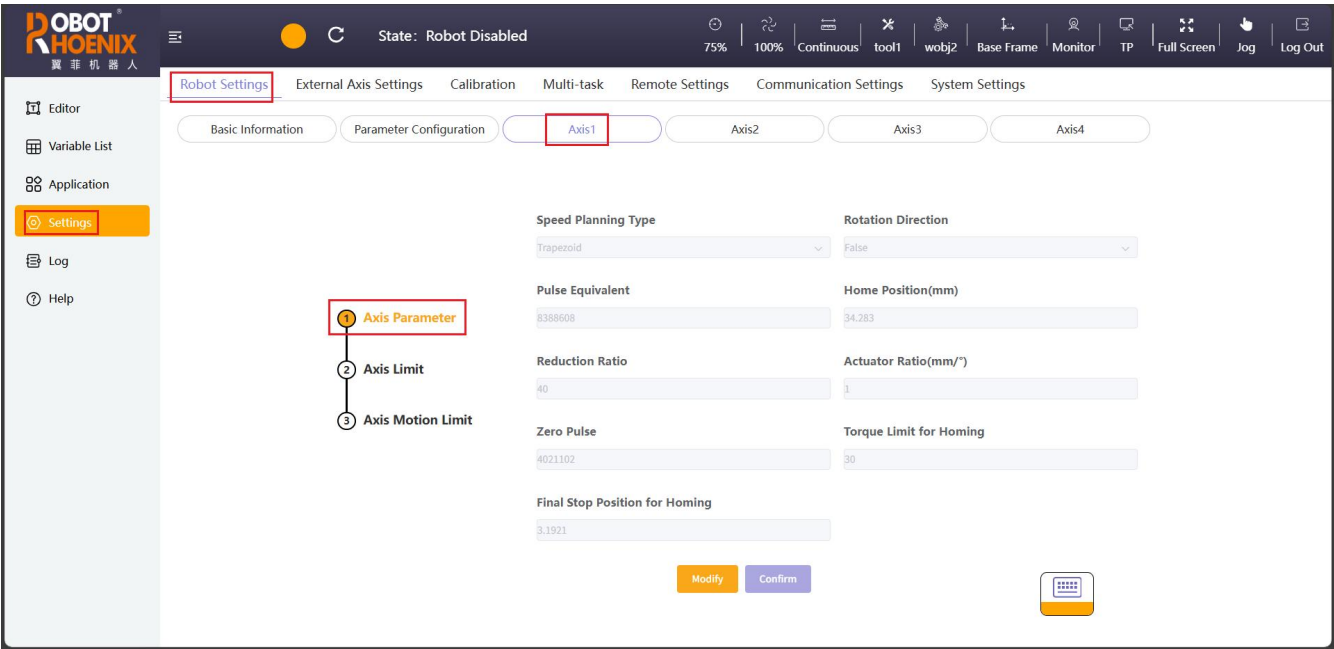
Items	Description
Enable Singularity Easing	False: Disabled True: Enabled
Axis 2 Easing Range Max	Sets the maximum threshold for Axis 2 to enter the easing zone. Easing will not be applied when this value is exceeded.
Axis 2 Easing Range Min	Sets the minimum threshold for Axis 2 to enter the easing zone. Easing will not be applied when this value is below the threshold.
Z-Axis Easing Coefficient	Z-axis performance scaling coefficient, valid range (0, 1].
Orientation Easing Coefficient	Rz-axis performance scaling coefficient, valid range (0, 1].
Axis 3 Easing Coefficient	Axis 3 performance scaling coefficient, valid range (0, 1].
Axis 4 Easing Coefficient	Axis 4 performance scaling coefficient, valid range (0, 1].

2.2.3. Axis Parameter Configuration

2.2.3.1. Axis Parameters

Operation method

Click "Settings" - "Robot Settings" - "Axis 1" - "Axis Parameters" in the left navigation bar to enter the axis parameter configuration interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



Description

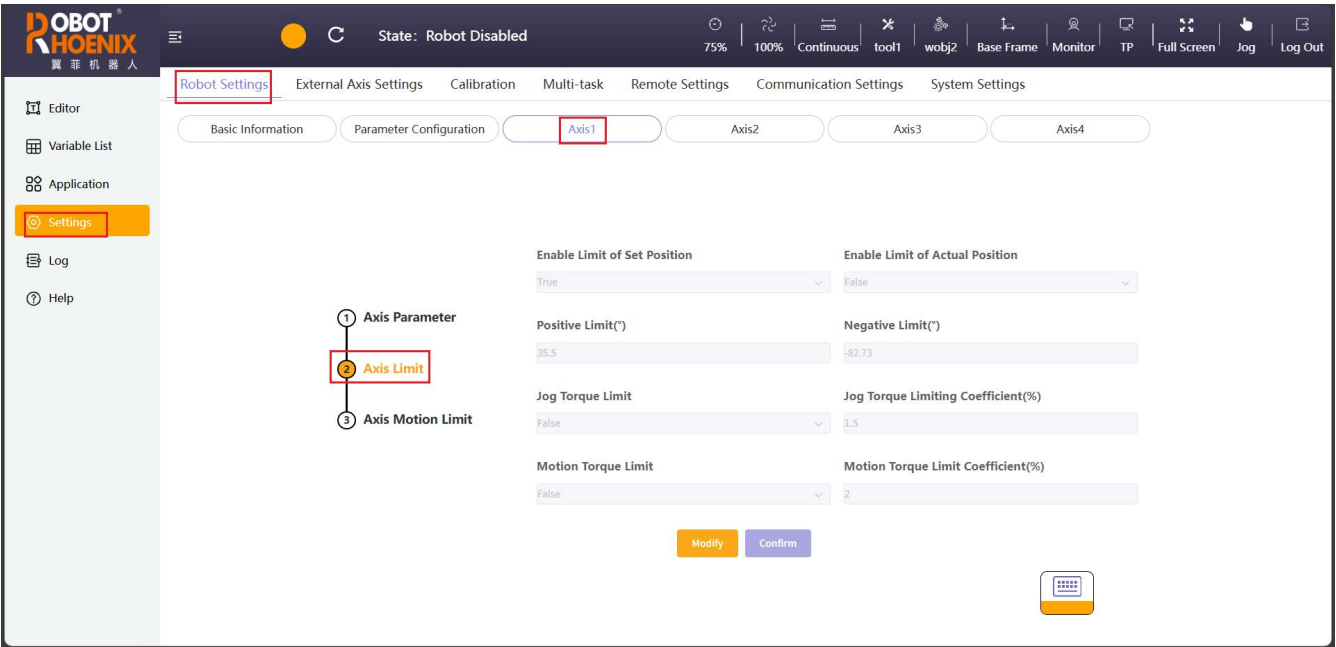
Items	Description
Speed planning method	Supports Trapezoid and Double_S_Curve
Rotation direction	TRUE/FALSE, TRUE: forward rotation; FALSE: reverse rotation
Pulse equivalent	The change in the motor encoder value when the motor shaft rotates one circle
Mechanical position corresponding to zero point	Default is 0
Reduction ratio	The ratio of the motor's instantaneous rotational speed to the instantaneous speed at the shaft end.
Execution end reduction ratio	The amount of workpiece movement per rotation of the motor shaft end, unit: mm/°

Zero point pulse	Pulse equivalent when calibrating zero point
Homing torque limit	Unit: %
Homing final position	Unit: °

2.2.3.2. Axis Constraints

Operation method

Click "Settings" - "Robot Settings" - "Axis 1" - "Axis Constraints" in the left navigation bar to enter the axis constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



Description

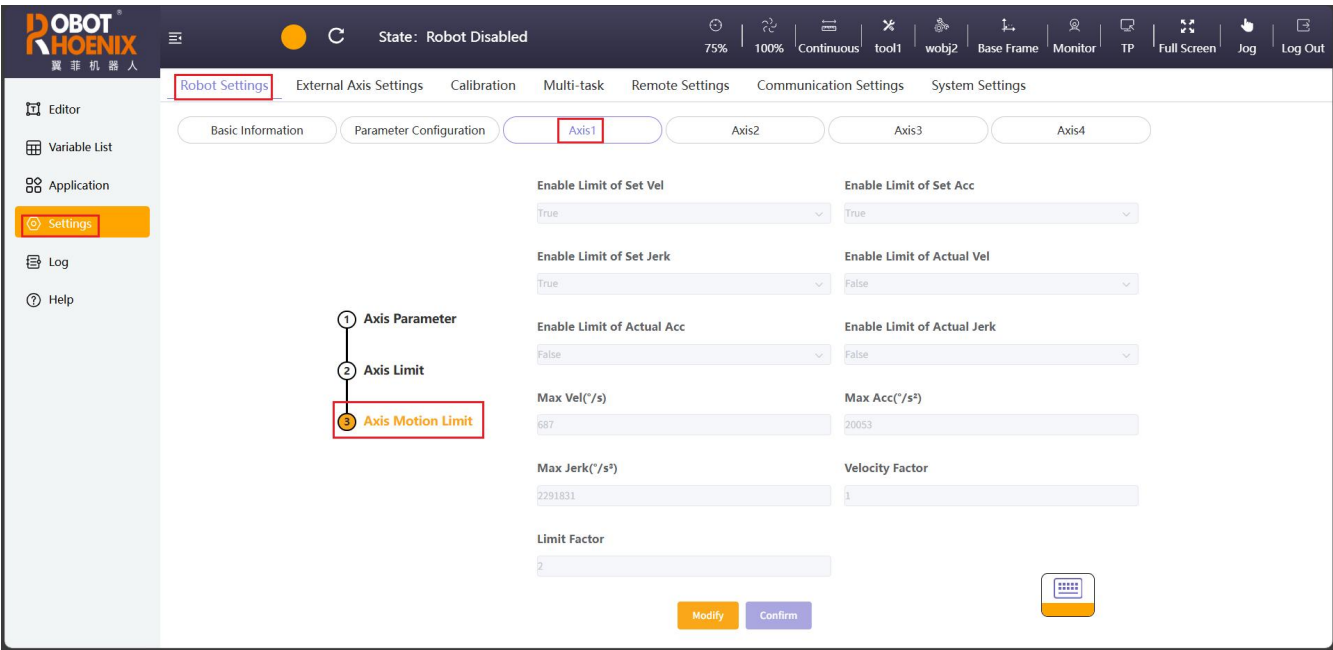
Items	Description
Planned position limit	False: Disabled True: Enabled
Actual position limit	False: Disabled True: Enabled
Positive limit (unit: °)	Default values have been set at the factory.
Negative limit (unit: °)	Default values have been set at the factory.
Jog Torque Limit	False: Disabled

	True: Enabled
Jog Torque Limit Coefficient	Unit: %
Motion Torque Limit	False: Disabled True: Enabled
Motion Torque Limit Coefficient	Unit: %

2.2.3.3. Axis Motion Constraints

Operation method

Click "Settings" - "Robot Settings" - "Axis 1" - "Axis Motion Constraints" in the left navigation bar to enter the axis motion constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



Description

Items	Description
Planned speed/acceleration/jerk limit	False: Disabled True: Enabled
Actual speed/acceleration/jerk limit	False: Disabled True: Enabled
Maximum speed/acceleration/jerk	Default values have been set at the factory.

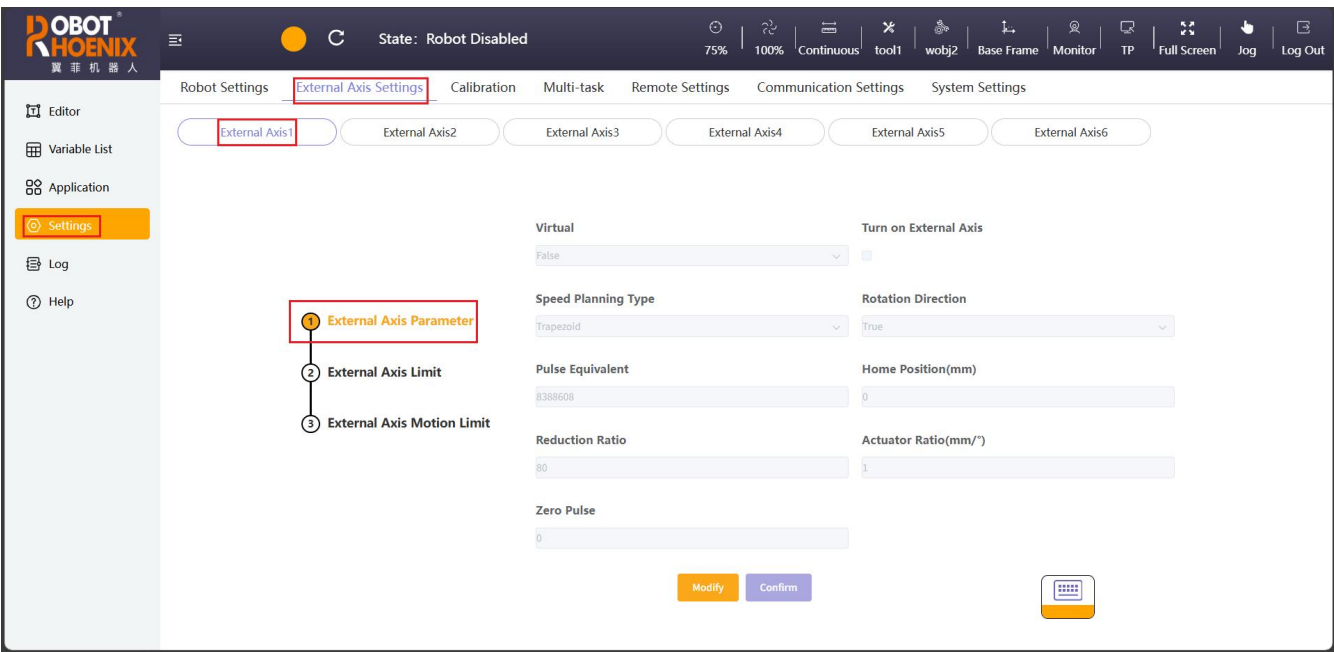
Speed coefficient	The default value is 1, and the valid range is (0,1].
Limit coefficient	The default value is 2, and the valid range is (0,10].

2.3. External Axis Settings

2.3.1. Axis Parameters for External Axes

Operation method

Click "Settings" - "External Axis Settings" - "External Axis 1" - "External Axis Parameters" in the left navigation bar to enter the external axis parameter setting interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



Description

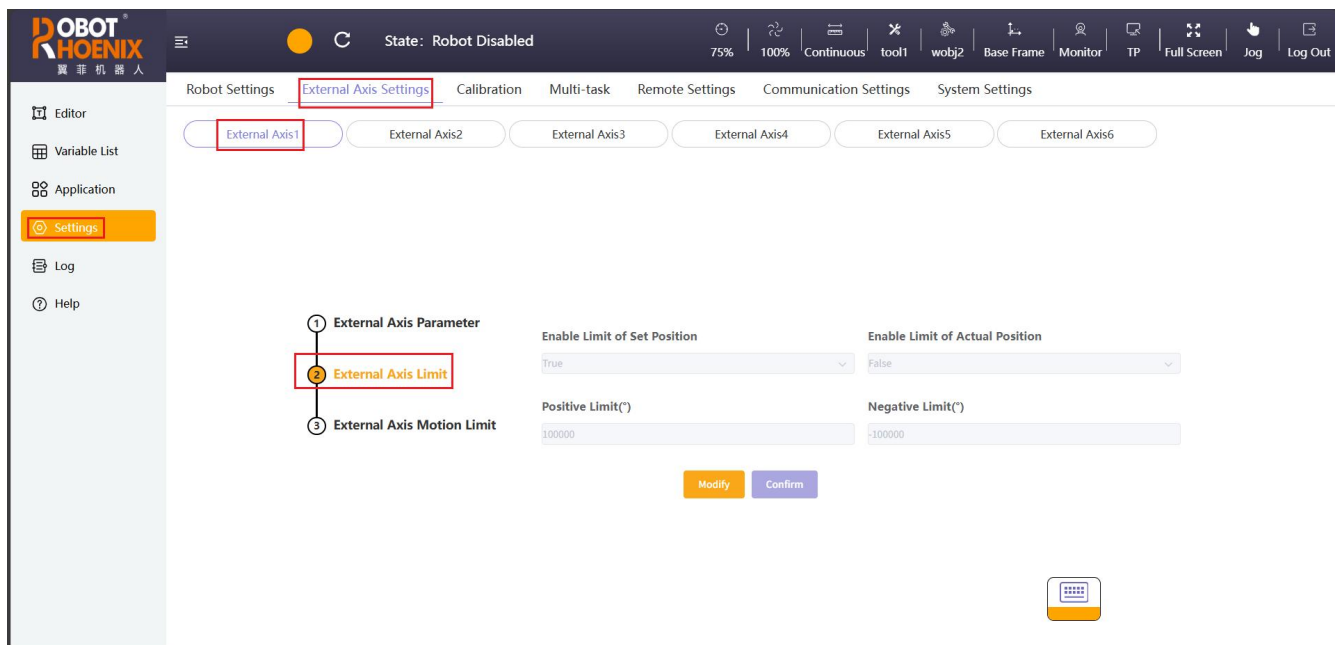
Items	Description
Whether to open the external axis	Open/Close
Whether it is a virtual axis	False: Real axis True: Virtual axis
Axis speed planning method	Supports Trapezoid and Double_S_Curve
Rotation direction	Change the positive direction of the axis

	rotation
Rotation direction	Change in number of pulses when the motor shaft rotates one revolution
Zero point corresponding mechanical position	Default is 0
Reduction ratio	The ratio of the motor's instantaneous rotational speed to the instantaneous speed at the shaft end.
Execution end reduction ratio	The amount of workpiece movement per degree of motor axis rotation, unit mm/°
Zero point pulse	After homing is completed, the pulse corresponding to the zero position is displayed, unit [pulse]

2.3.2. Axis Constraints for External Axes

Operation method

Click "Settings" - "External Axis Settings" - "External Axis 1" - "External Axis Constraints" in the left navigation bar to enter the external axis constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



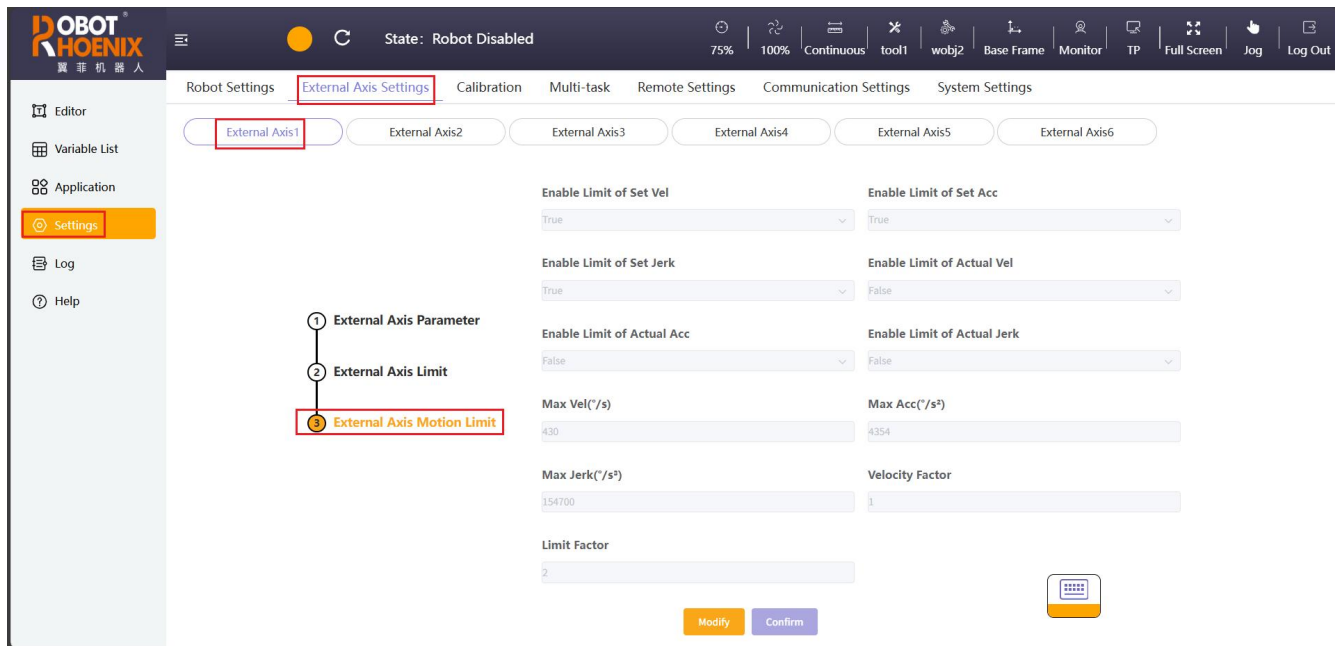
Description

Items	Description
Planned position limit	False: Disabled True: Enabled
Planned position limit	False: Disabled True: Enabled
Positive limit (unit: °)	Users can adjust according to the specific parameters of the robot.
Negative limit (unit: °)	Users can adjust according to the specific parameters of the robot.

2.3.3. Axis Motion Constraints for External Axes

Operation method

Click "Settings" - "External Axis Settings" - "External Axis 1" - "External Axis Motion Constraints" in the left navigation bar to enter the external axis motion constraint settings interface, click the "Modify" button to modify, and click "OK" after the modification is completed. The same applies to other axes. As shown below:



Description

Items	Description
Planned speed/acceleration/jerk limit	False: Disabled True: Enabled
Actual speed/acceleration/jerk limit	False: Disabled True: Enabled
Maximum speed/acceleration/jerk	The setting value must be greater than 0, and users can adjust it according to actual conditions.
Speed coefficient	The default value is 1, and the valid range is (0,1].
Limit coefficient	The default value is 2, and the valid range is (0,10].

2.4. Calibration

The tool coordinate system, work object coordinate system, and user coordinate system can be calibrated, and robot zero position calibration, external axis zero position calibration, and zero position offset can be performed.

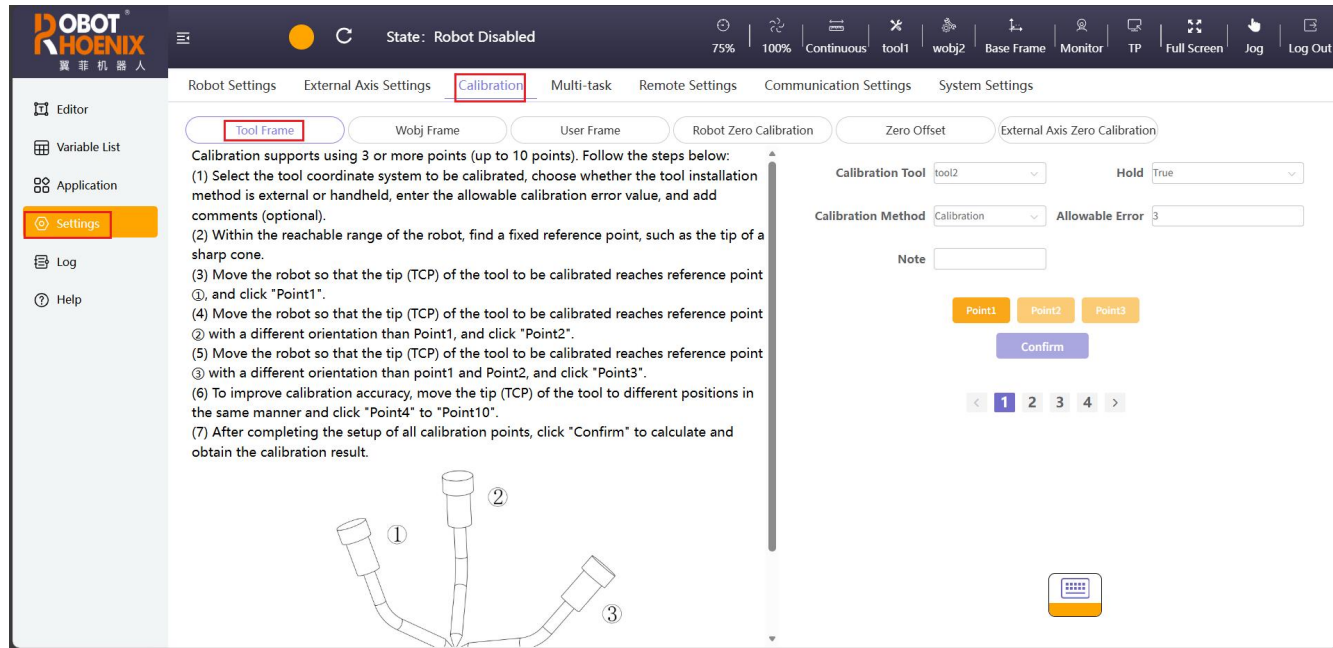
2.4.1. Tool Coordinate System Calibration

After installing tools such as the gripper, suction cup or welding gun at the end of the robot, the tool coordinate system needs to be calibrated before use to determine the relative relationship between the working position and orientation of the tool and the robot base coordinate system, so that the position and orientation of the tool end can be accurately controlled for work.

The default tool coordinate system is located at the center of the robot flange.

2.4.1.1. Operation method

As shown in the figure below, click "Settings" - "Calibration" - "Tool Coordinate System" in the left navigation bar to enter the tool coordinate system calibration interface for calibration.



The tool coordinate system number ranges from 1 to 16, where coordinate system 1 is the tool number used when the tool is not installed, and is also the default setting value of the tool number. Except for coordinate system 1, the other 15 coordinate systems can match the actual tool and be calibrated.

Tool calibration includes manual input, three-point method, four-point method, six-point method, and calibration.

For 6-axis robots, the three-point method, four-point method, and six-point method are used to calibrate the position and orientation of the tool coordinate system; for SCARA and Delta robots, the calibration method is used to calibrate the X and Y of the tool coordinate system.

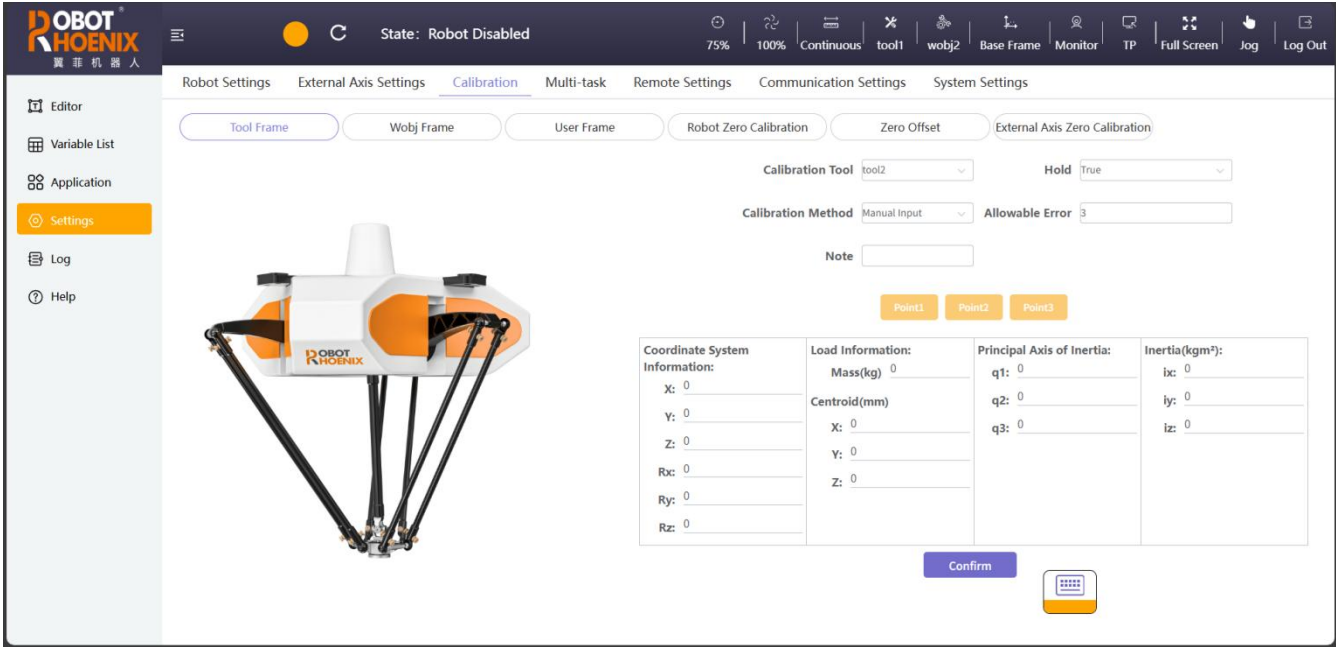
These methods are explained below.

Method	Features	Applicable scenarios
Manual input	Directly input coordinate system parameters	Known coordinate system parameters or modify the calibrated coordinate system.
Three-point method	Get the tool orientation through three-point alignment	Need to manually calibrate the orientation of the tool coordinate system, applicable to six-axis robots.
Four-point method	Get the tool position through four-point alignment	Need to manually calibrate the position of the tool coordinate system, applicable to six-axis robots.

Six-point method	Get the tool position and orientation through six-point alignment	Need to manually calibrate the position and orientation of the tool coordinate system, applicable to six-axis robots.
Calibration	Get the tool position through multi-point alignment	Applicable to SCARA and DELTA robots.

2.4.1.1.1. Manual Input

If you already know the value of the tool coordinate system or need to modify the calibrated coordinate system, you can select "Manual Input", directly enter the coordinate system information and click OK.

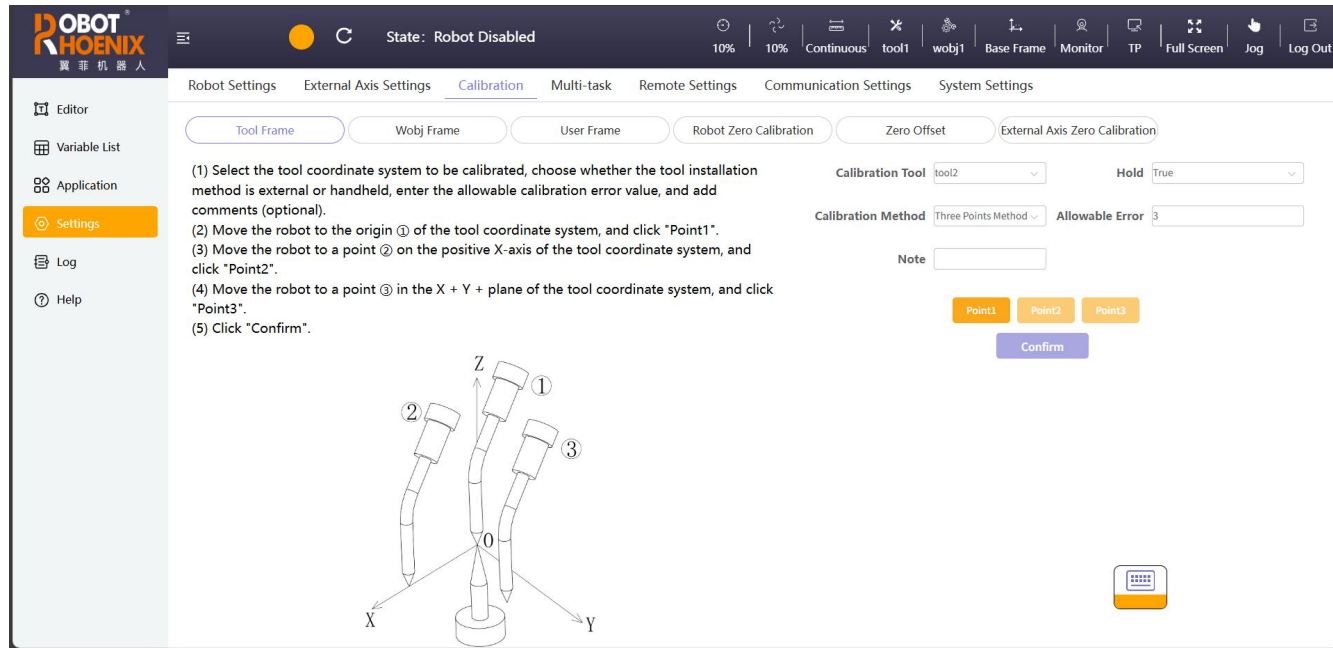


Note:

The tool coordinate system information of SCARA robot and delta robot can only contain X and Y, and other values must be 0.

2.4.1.1.2. Three-Point Method

This method is suitable for calibrating the orientation vector of the TCP at the end of the 6-axis robot tool.



After installing the tool at the end of the robot, follow the steps below:

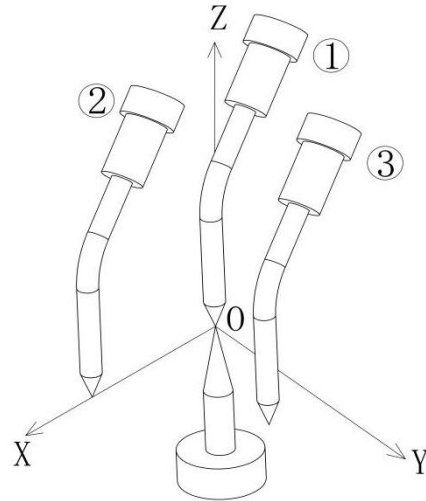
Step 1: Select the tool coordinate system to be calibrated, select the "three-point method" calibration method, select whether the tool is installed externally or handheld, fill in the calibration allowable error value, and fill in the comments (optional).

Step 2: Move the robot to the origin of the tool coordinate system (Point ①), then click "Get Point 1."

Step 3: Move the robot to a point on the positive X-axis of the tool coordinate system (point ②), then click "Get Point 2".

Step 4: Move the robot to a point within the X+Y+ plane of the tool coordinate system (point ③), then click "Get Point 3".

Step 5: Click "OK".

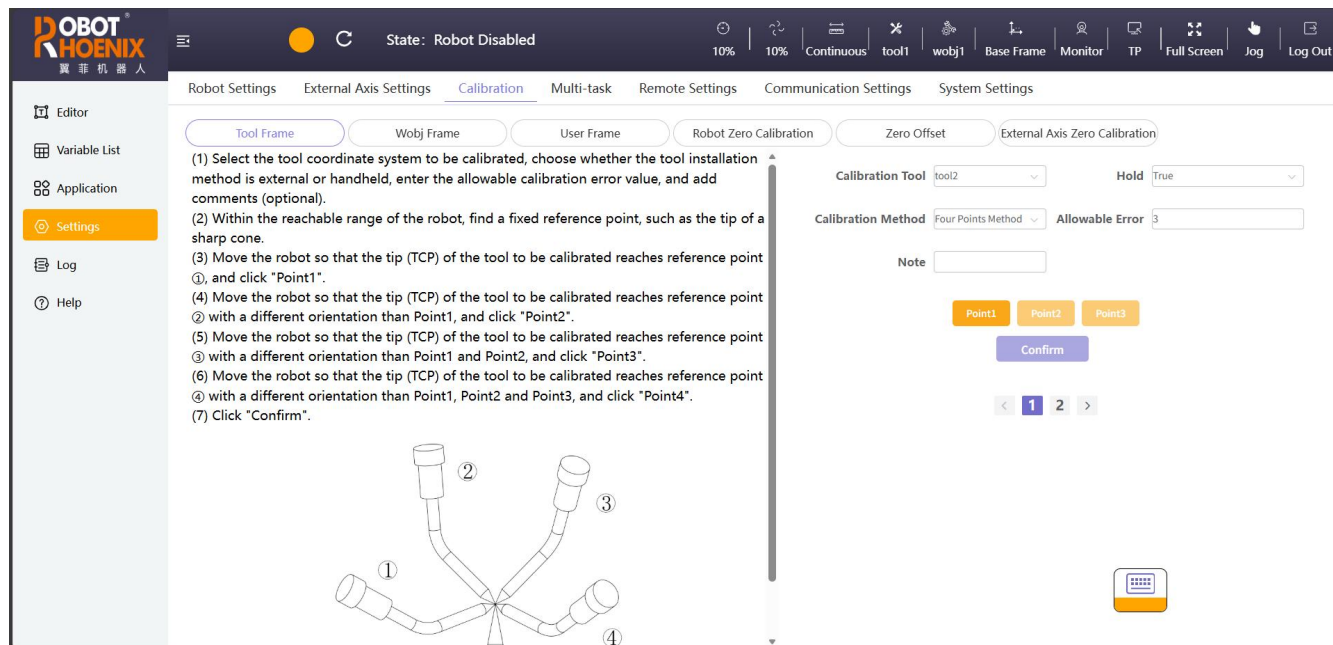


Note:

When the “Three-point method” is used to determine the tool orientation, the three position points can only be taught by translation motion in Cartesian space, and cannot be taught by rotation motion with any orientation.

2.4.1.1.3. Four-Point Method

This method is suitable for calibrating the position offset value of the TCP at the end of a 6-axis robot tool.



After installing the tool on the end of the robot, follow the steps below:

Step 1: Select the tool coordinate system to be calibrated, select the "four-point method" calibration method, select whether the tool is installed externally or handheld, fill in the calibration allowable error value, and fill in the comments (optional).

Step 2: Find a fixed reference point within the reach of the robot, such as the top of a pointed cone.

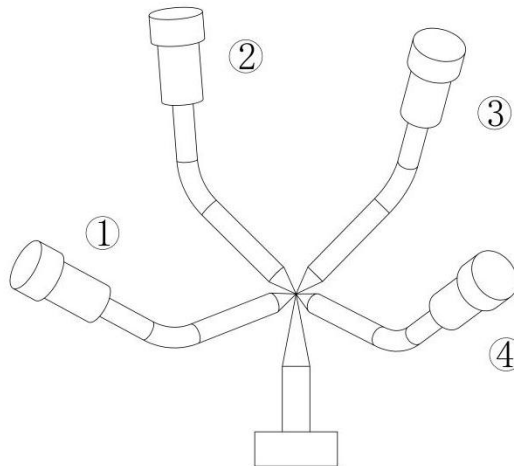
Step 3: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ①, then click "Get Point 1."

Step 4: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ② with an orientation different from Point 1, then click "Get Point 2."

Step 5: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ③ with an orientation different from Points 1 and 2, then click "Get Point 3."

Step 6: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ④ with an orientation different from Points 1, 2 and 3, then click "Get Point 4."

Step 7: Click "OK".



Note:

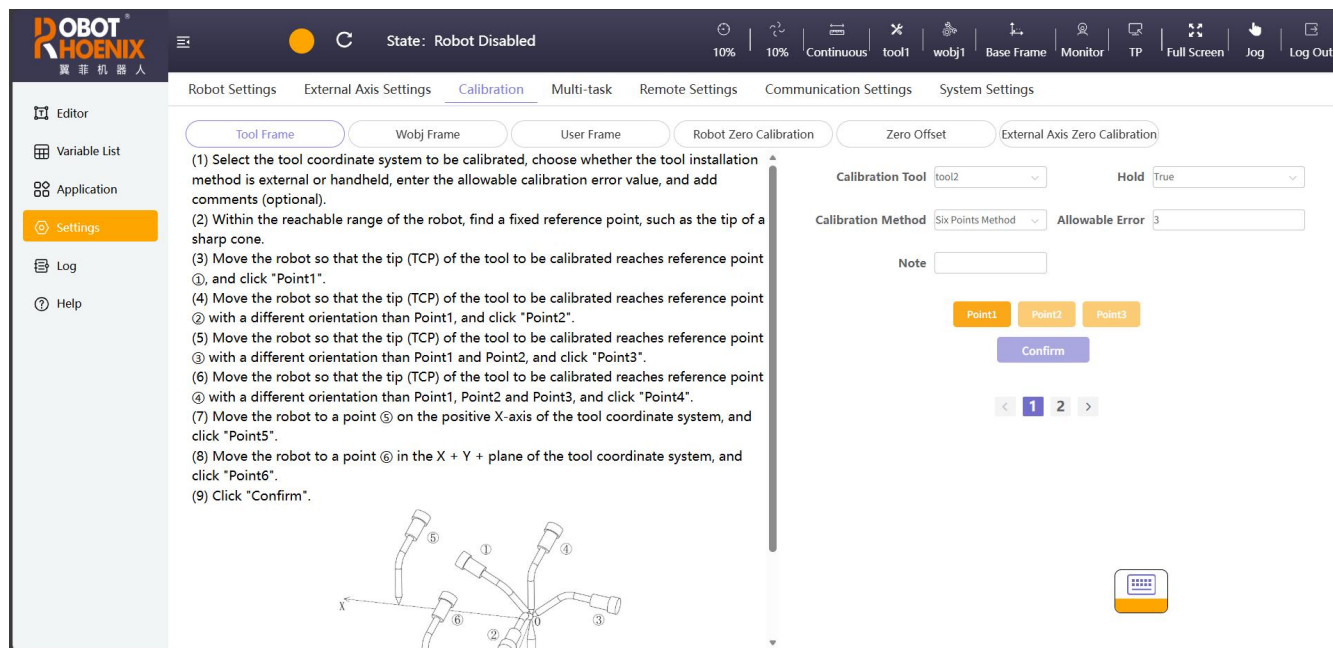
The "four-point method" can only determine the position offset of the tool tip point TCP relative to the robot end flange mounting surface. When the user needs to teach and determine the tool orientation component, it is necessary to use the "three-point method" additionally, or directly use the "six-point method". If two or more identical position points are recorded in the "four-point method", the calibration cannot be successful. The positions of the four reference points should be as dispersed as possible to make the calculated TCP point as accurate as

possible.

2.4.1.1.4. Six-Point Method

This method is suitable for calibrating the TCP position offset and orientation vector of the tool end of a 6-axis robot.

The "three-point method" can only calculate the TCP orientation vector of the end of the 6-axis robot tool, but cannot calculate the position offset value of the TCP at the end of the tool; the "four-point method" can only calculate the position offset value of the TCP at the end of the 6-axis robot tool, but cannot calculate the orientation vector of the tool. The "six-point method" is actually a comprehensive use of the "three-point method" and the "four-point method". The first four points recorded in the "six-point method" use the "four-point method" to calculate the position offset of the TCP at the end of the tool, and the last three points use the "three-point method" to calculate the TCP orientation vector of the tool.



After installing the tool on the end of the robot, follow the steps below:

Step 1: Select the tool coordinate system that needs to be calibrated, select the "six-point method" calibration method, select whether the tool is installed externally or handheld, fill in the calibration allowable error value, and fill in comments (optional).

Step 2: Find a fixed reference point within the reach of the robot, such as the top of a pointed cone.

Step 3: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference

Point ①, then click “Get Point 1.”

Step 4: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ② with an orientation different from Point 1, then click “Get Point 2.”

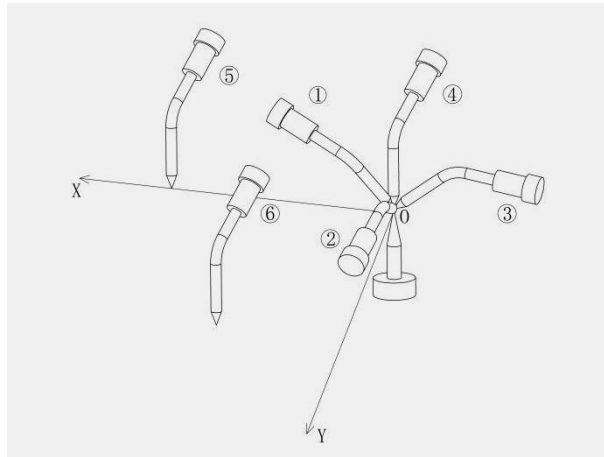
Step 5: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ③ with an orientation different from Points 1 and 2, then click “Get Point 3.”

Step 6: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ④ with an orientation different from Points 1, 2 and 3, then click “Get Point 4.”

Step 7: Move the robot so that the tip of the tool to be measured (TCP) reaches a point on the positive X-axis of the tool coordinate system (point ⑤), then click “Get Point 5”.

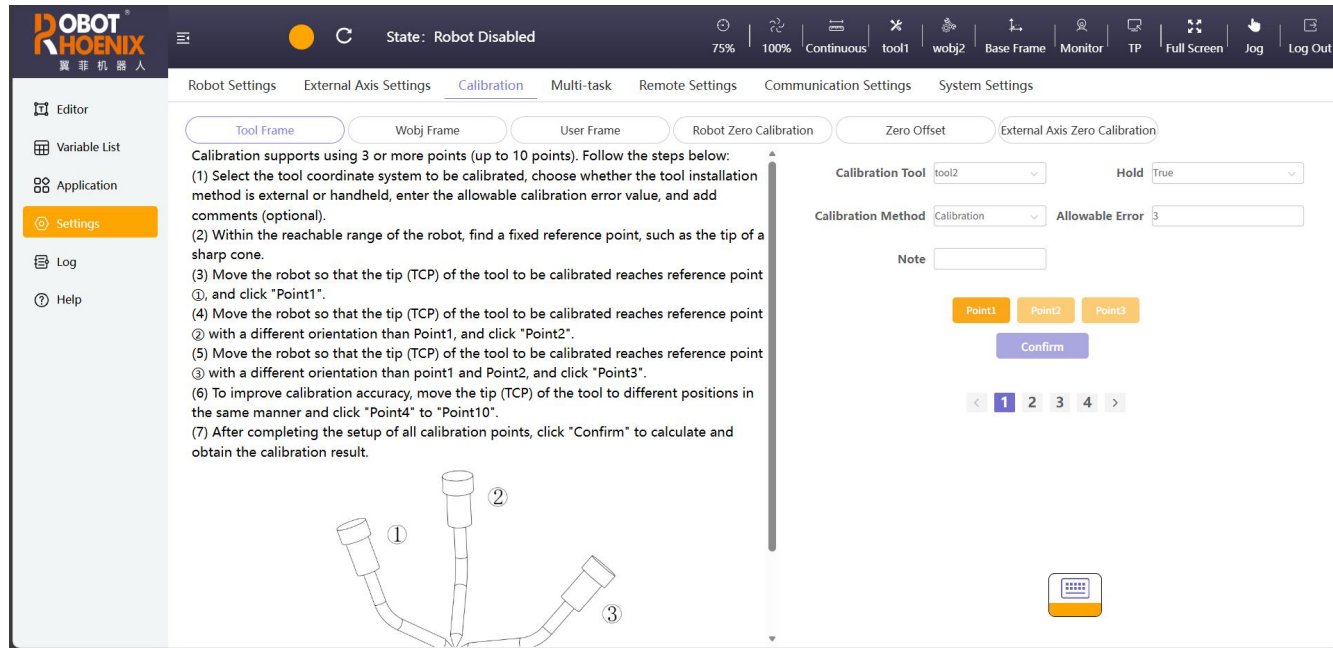
Step 8: Move the robot so that the tip of the tool to be measured (TCP) reaches a point within the X+Y+ plane of the tool coordinate system (point ⑥), then click “Get Point 6”.

Step 9: Click "OK".



2.4.1.1.5. Calibration

This method is applicable for calibrating the tool coordinate system of SCARA robots and Delta robots. The function supports tool coordinate system calibration using three or more points (up to ten points) to improve calibration accuracy.



After installing the tool on the end of the robot, follow the steps below:

Step 1: Select the tool coordinate system that needs to be calibrated, select the "six-point method" calibration method, select whether the tool is installed externally or handheld, fill in the calibration allowable error value, and fill in comments (optional).

Step 2: Find a fixed reference point within the reach of the robot, such as the top of a pointed cone.

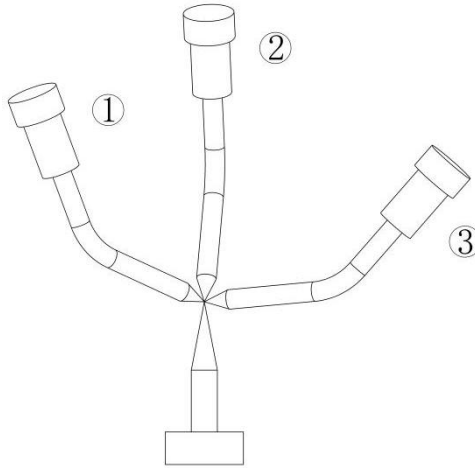
Step 3: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ①, then click “Get Point 1.”

Step 4: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ② with an orientation different from Point 1, then click “Get Point 2.”

Step 5: Move the robot so that the tip of the tool to be measured (TCP) reaches Reference Point ③ with an orientation different from Points 1 and 2, then click “Get Point 3.”

Step 6: If higher calibration accuracy is required, continue moving the robot so that the tip of the tool to be measured (TCP) reaches different positions, clicking “Get Point 4” through “Get Point 10” as needed.

Step 7: After setting all calibration points, click “Confirm” to calculate and obtain the calibration result.

**Note:**

When moving the robot's end-effector TCP to reach the reference points with different orientations, try to make the orientation variations between the taught points as large as possible.

2.4.1.2. Operation Results

To verify whether the tool calibration results are accurate, you can use the following method, the steps are as follows:

Step 1: Click the Jog Coordinate System button () on the top toolbar to switch to the base coordinate system.

Step 2: Click the Tool Coordinate System button () on the top toolbar and select the tool coordinate system that has been calibrated.

Step 3: Align the tool with a fixed reference point (such as a sharp corner of an object or a calibration cone).

Step 4: Jog the robot to change the tool orientation ($R_z+/-$) and observe whether the tool end point is always near the selected reference point. (The smaller the deviation, the higher the accuracy)

2.4.2. Work Object Coordinate System Calibration

A coordinate system defined on the workpiece. A well-defined work object coordinate system

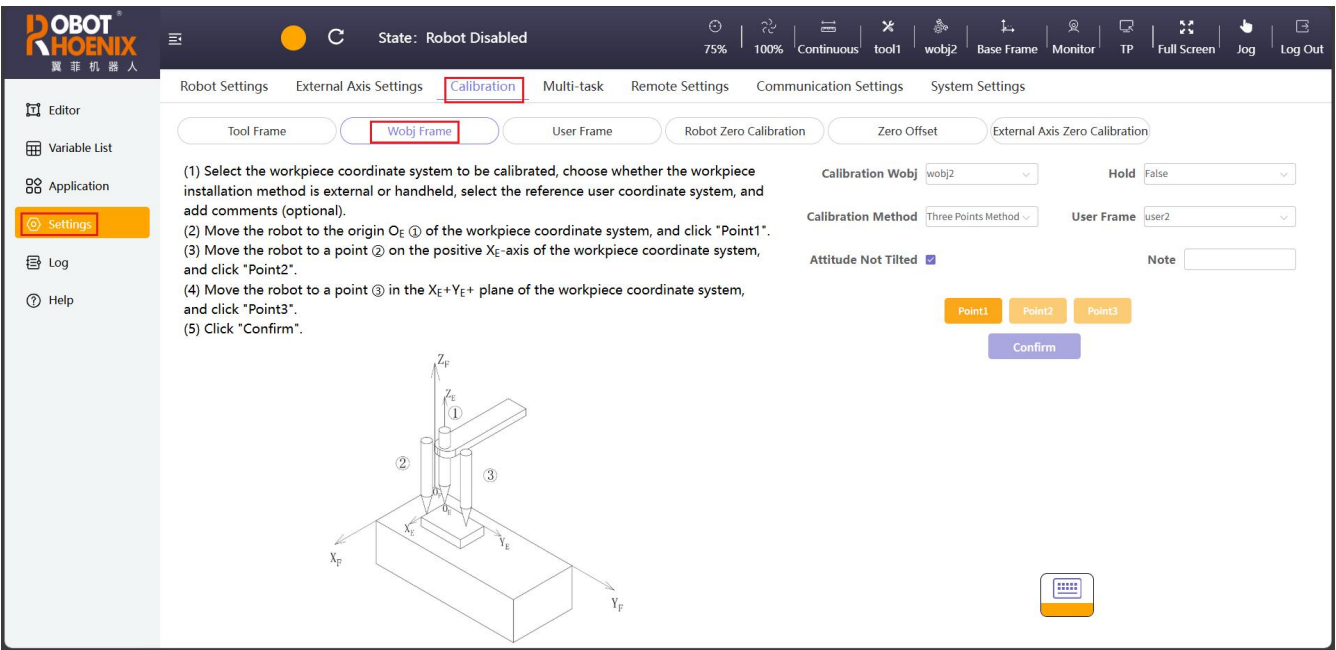
can greatly reduce programming complexity and improve program reusability.

The work object coordinate system is defined relative to the user coordinate system. When calibrating the work object coordinate system, you need to select a reference user coordinate system.

Before calibrating the workpiece, you need to calibrate a tool first, and then use the TCP of the tool to calibrate the work object coordinate system. For ease of operation, it is recommended to use a tool with a tip.

2.4.2.1. Operation Method

As shown in the figure below, click "Settings" - "Calibration" - "Work Object Coordinate System" in the left navigation bar to enter the work object coordinate system calibration interface for calibration.



The work object coordinate system number range is 1-16, where coordinate system 1 is the workpiece number used when the workpiece is not installed, and is also the default setting value of the workpiece number. Except for coordinate system 1, the other 15 coordinate systems can match the actual workpiece and be calibrated.

Workpiece calibration includes manual input and three-point method.

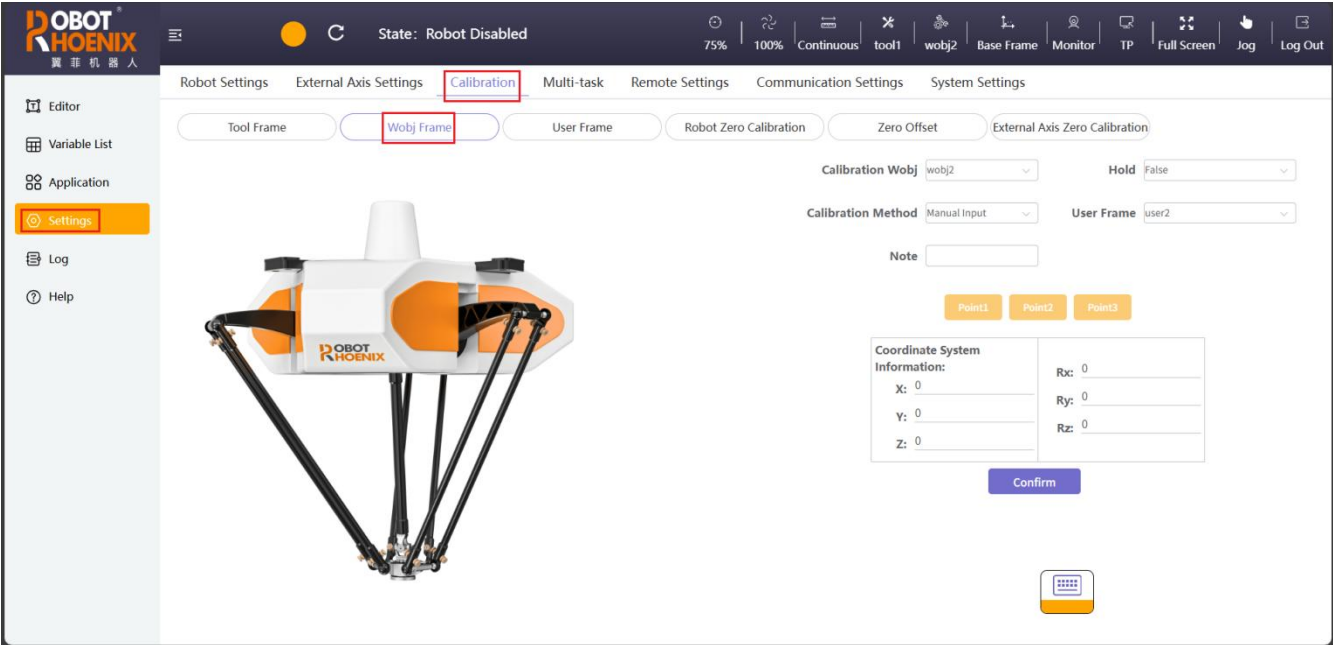
These methods are explained below.

Method	Features	Applicable scenarios
--------	----------	----------------------

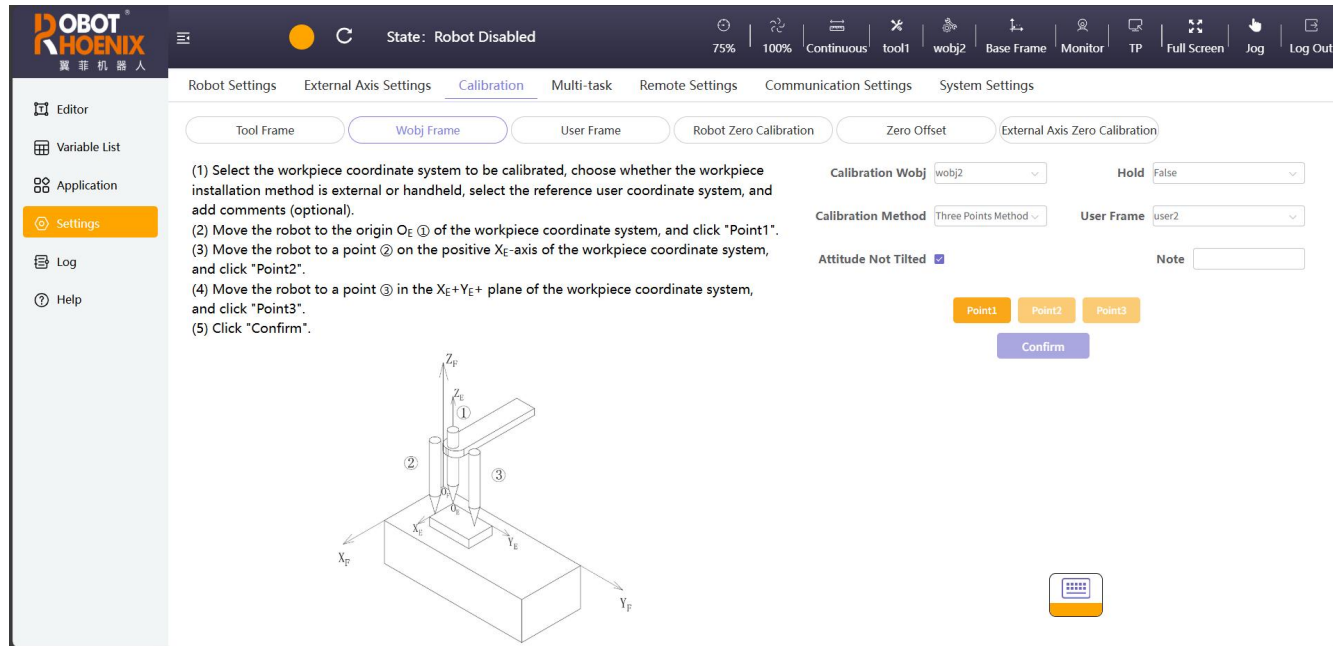
Manual input	Directly input coordinate system parameters	Known coordinate system parameters or modify the calibrated coordinate system.
Three-point method	Calibrate the work object coordinate system by three points	The value of the work object coordinate system needs to be manually calibrated.

2.4.2.1.1. Manual Input

If you already know the value of the work object coordinate system or need to modify the calibrated coordinate system value, you can select "Manual Input", directly enter the value and click OK.



2.4.2.1.2. Three-Point Method



Follow the steps below:

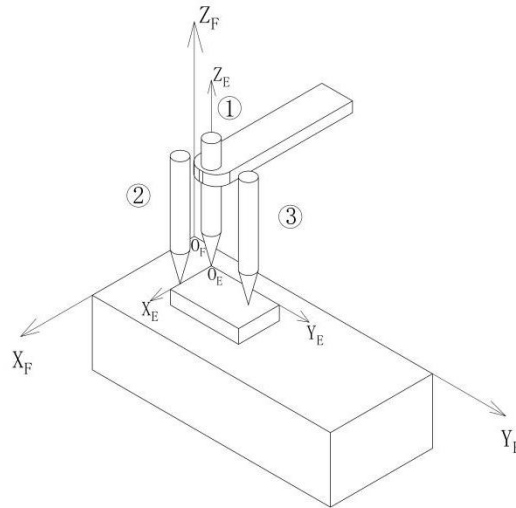
Step 1: Select the work object coordinate system to be calibrated, select the "three-point method" calibration method, select whether the workpiece installation method is external or handheld, select the reference user coordinate system, and fill in the comments (optional).

Step 2: Move the robot to the origin O_E ① of the work object coordinate system and click "Get Point 1."

Step 3: Move the robot to a point ② along the positive X_E -axis direction of the work object coordinate system, and click "Get Point 2."

Step 4: Move the robot to a point ③ within the $X_E + Y_E +$ plane of the work object coordinate system, and click "Get Point 3."

Step 5: Click "OK".

**Note:**

If you use a calibrated tool to calibrate the work object coordinate system, you need to select the corresponding tool number in the toolbar-tool coordinate system ( tool1).

When “No Tilt Orientation” is checked, the second and third points will be forced to use the Z-value of the first point, ensuring that the calibrated coordinate system’s $X_E Y_E$ plane is parallel to the selected user coordinate system’s $X_F Y_F$ plane.

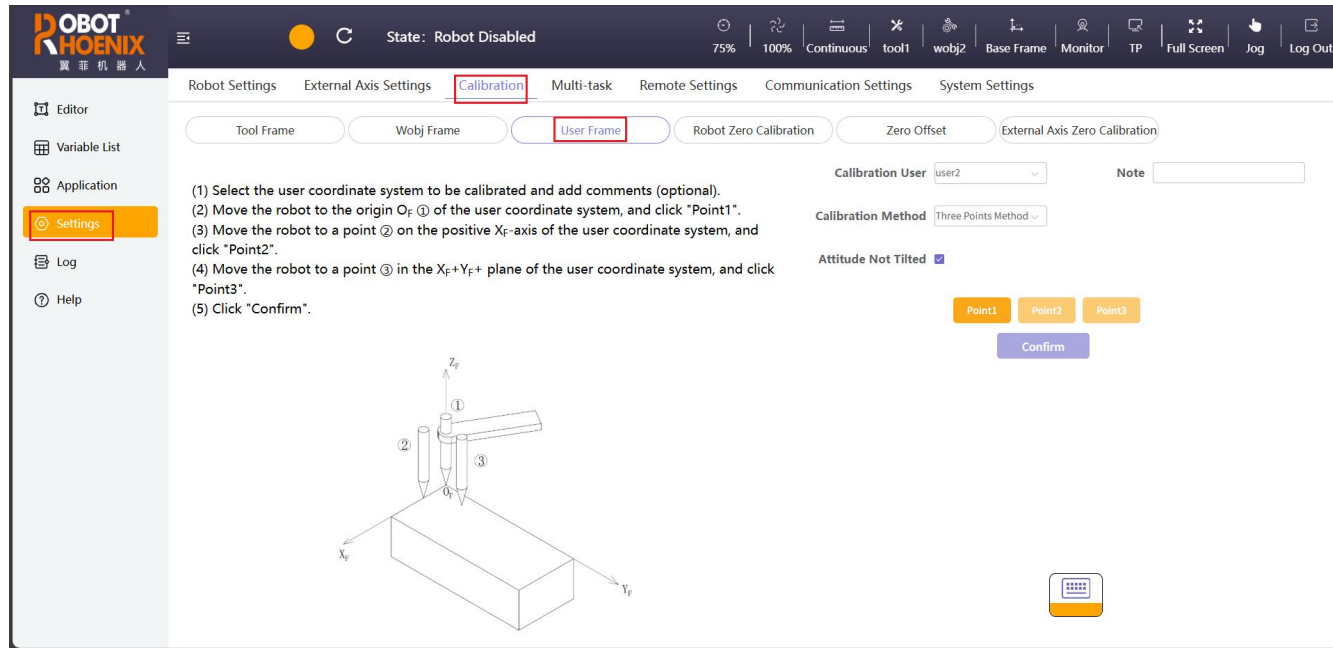
2.4.3. User Coordinate System Calibration

The user coordinate system serves as a reference system when defining the work object coordinate system and is not used alone.

The methods used to calibrate the user coordinate system are also manual input, three-point method and rotation method. The operation steps are basically similar to those of calibrating the work object coordinate system.

2.4.3.1. Operation method

As shown in the figure below, click "Settings" - "Calibration" - "User Coordinate System" in the left navigation bar to enter the user coordinate system calibration interface for calibration.



The user coordinate system number range is 1-16, among which coordinate system 1 is the default user coordinate system and cannot be modified. Except for coordinate system 1, the other 15 coordinate systems can match the actual user and be calibrated.

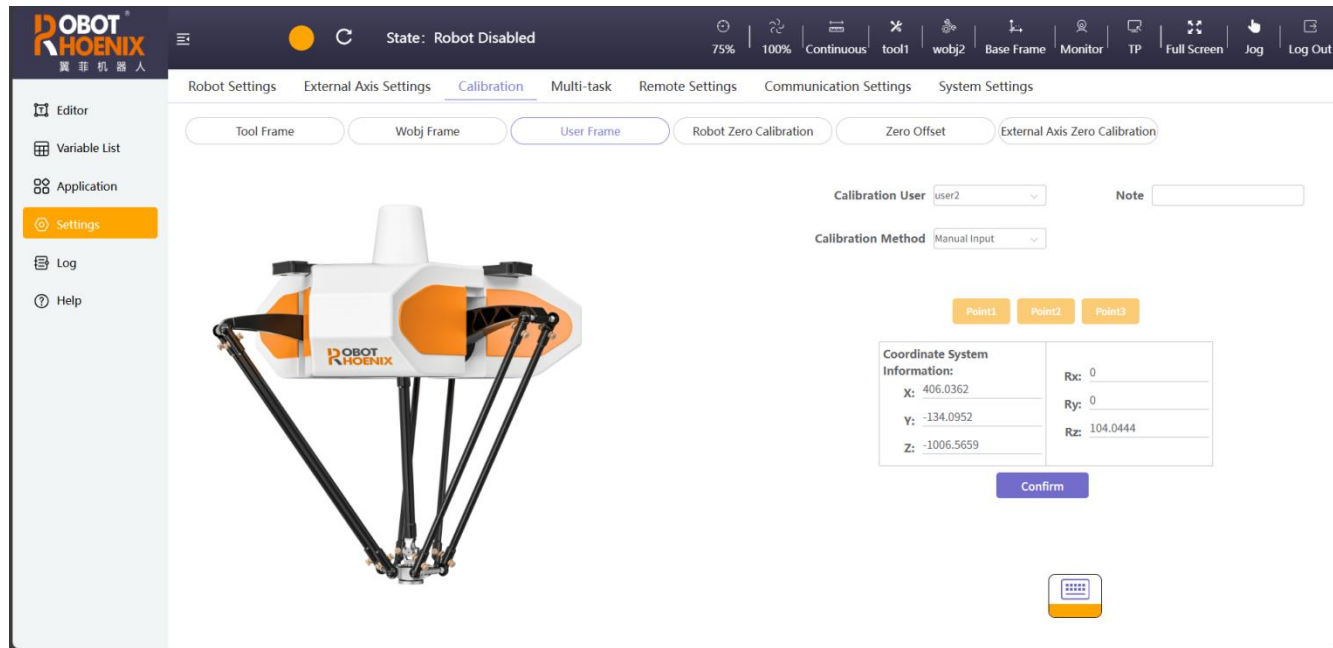
User calibration includes manual input, three-point method, and rotation method.

These methods are described below.

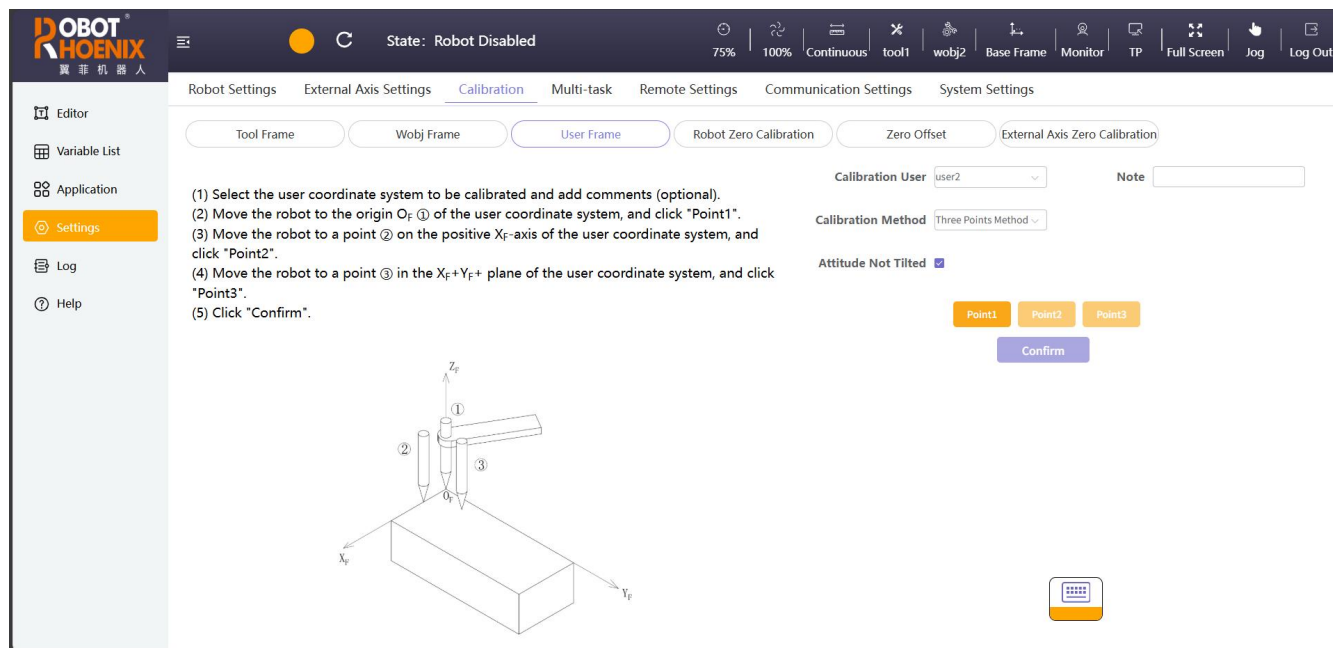
Method	Features	Applicable scenarios
Manual input	Directly input coordinate system parameters	Known coordinate system parameters or modify the calibrated coordinate system
Three-point method	Calibrate the user coordinate system by three points	The value of the user coordinate system needs to be calibrated manually.
Rotation method	Mark points on the turntable, rotate the turntable, and teach points	With a turntable, and the user coordinate system is located at the center of the turntable.

2.4.3.1.1. Manual Input

If you already know the value of the user coordinate system or need to modify the calibrated coordinate system value, you can select "Manual Input", directly enter the value and click OK.



2.4.3.1.2. Three-Point Method



Follow the steps below:

Step 1: Select the user coordinate system to be calibrated, select the "three-point method" calibration method, and fill in the comments (optional).

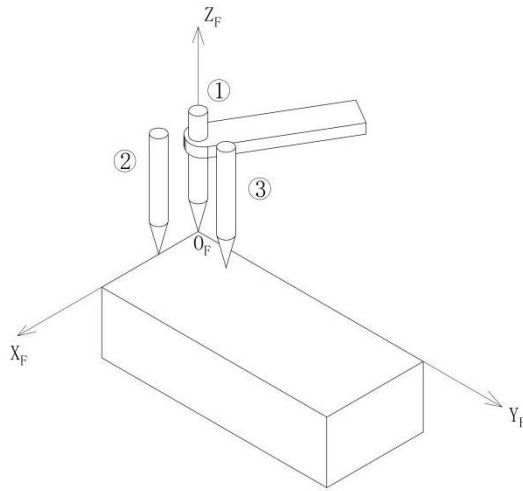
Step 2: Move the robot to the origin O_F ① of the user coordinate system, and click "Get Point

1".

Step 3: Move the robot to a point ② on the positive direction of the X_F -axis of the user coordinate system, and click "Get Point 2".

Step 4: Move the robot to a point ③ within the $X_F + Y_F +$ plane of the work object coordinate system, and click "Get Point 3."

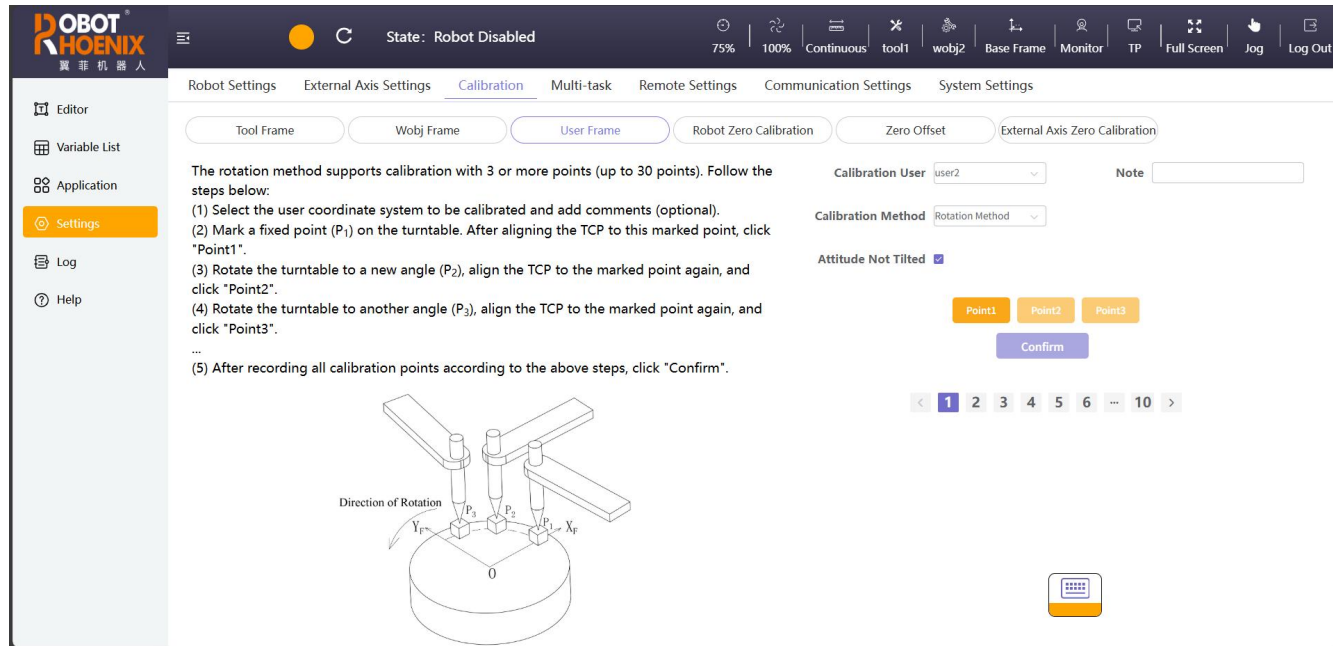
Step 5: Click "OK".



Note:

When "No Tilt Orientation" is checked, the second and third points will be forced to use the Z-value of the first point, ensuring that the calibrated coordinate system's $X_F Y_F$ plane is parallel to the selected base coordinate system's XY plane.

2.4.3.1.3. Rotation Method



The rotation method supports calibration using 3 or more points (up to 30 points). Follow the steps below to perform the calibration:

Step 1: Select the user coordinate system to be calibrated, select the “rotation method” calibration method, and fill in the comments (optional).

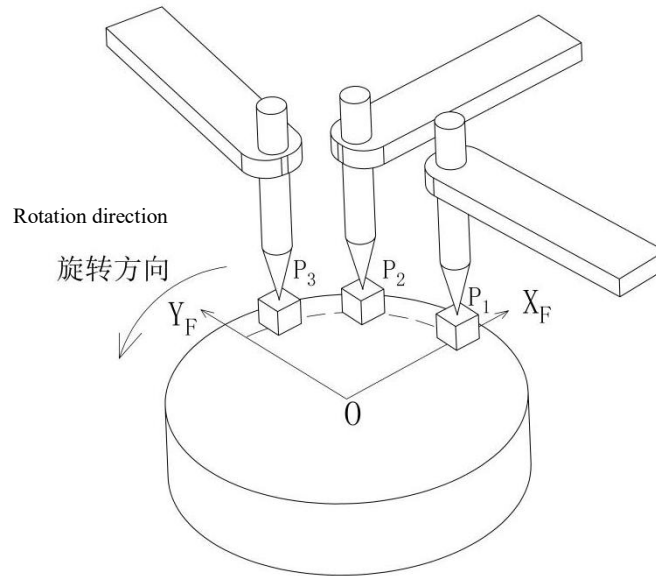
Step 2: Mark a fixed point (P_1) on the turntable, use the TCP to align the marked point, and click “Get Point 1”.

Step 3: Rotate the turntable by an angle (P_2), use the TCP to align the marked point again, and click “Get Point 2”.

Step 4: Rotate the turntable by an angle (P_3) again, use the TCP to align the marked point again, and click “Get Point 3”.

...

Step 5: Repeat the above steps to capture all calibration points, then click “OK.”

**Note:**

P_1 is a point on the positive direction of the X_F -axis of the coordinate system to be calibrated, and P_2 is a point within the $X_F + Y_F$ plane of the coordinate system to be calibrated.

When “No Tilt Orientation” is checked, the second and third points will be forced to use the Z-value of the first point, ensuring that the calibrated coordinate system’s $X_F Y_F$ plane is parallel to the base coordinate system’s XY plane.

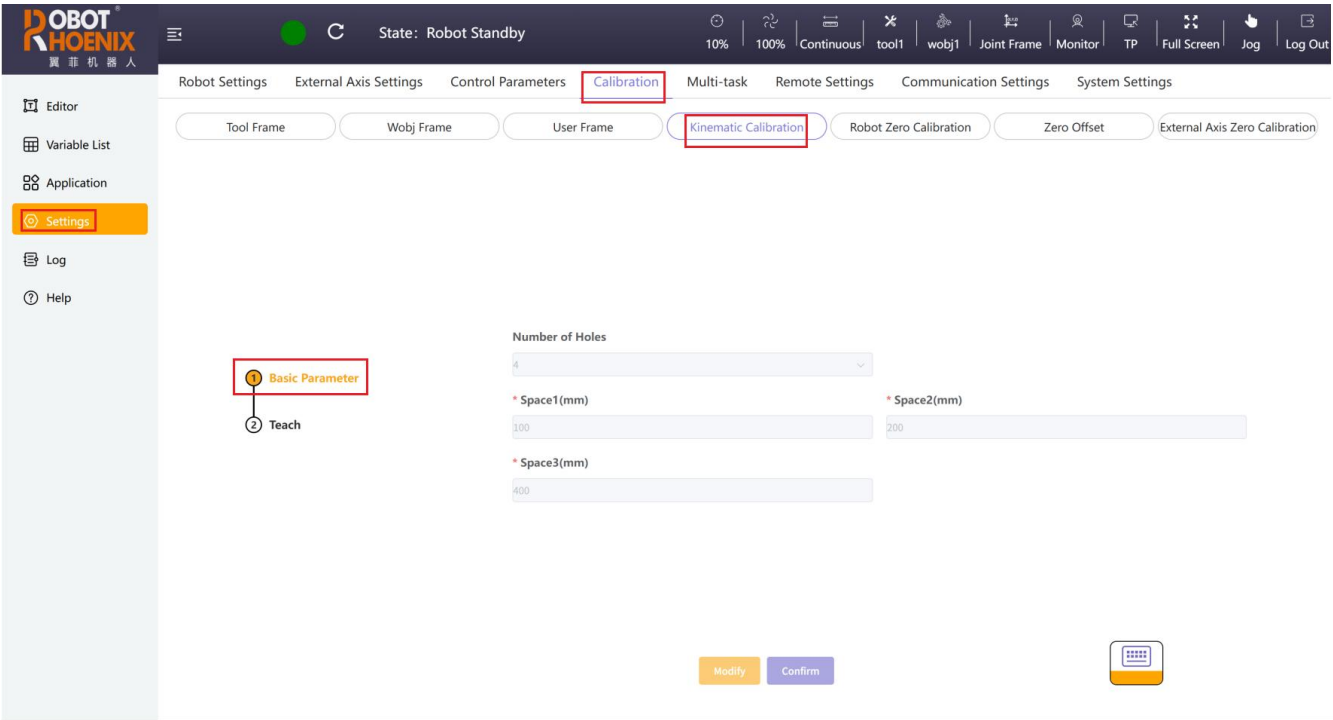
2.4.4. Kinematics Calibration

Gorilla Studio robot control system provides kinematics calibration only for SCARA robots.

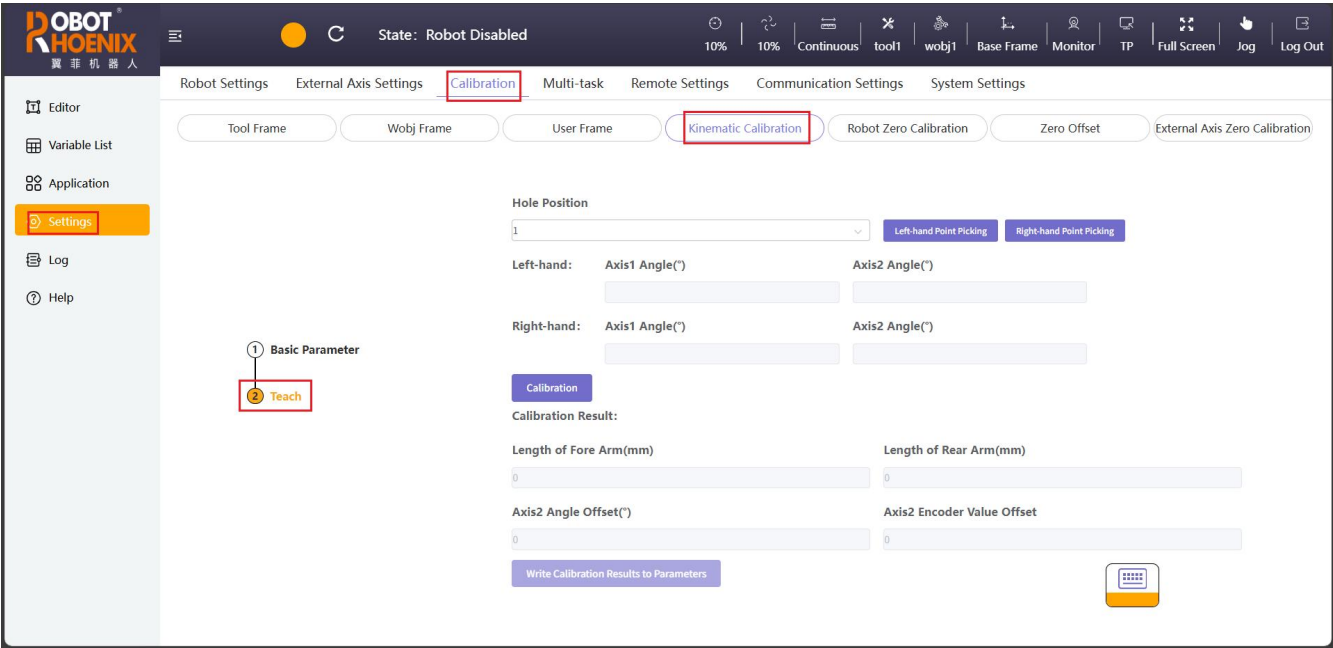
Kinematics calibration is used to correct discrepancies between the robot model and the actual robot, thereby improving the robot’s positioning accuracy within its workspace.

Operation method

As shown below, click “Settings” → “Calibration” → “Kinematics Calibration” on the left navigation panel to enter the Kinematics Calibration interface. In the basic parameters, you can select the number of holes (1–4) and specify the corresponding spacing lengths.



For each hole, sequentially teach the robot's left-hand and right-hand poses. After teaching the poses for all holes, click the "Calibrate" button. The system will perform calculations based on the collected data. The calculation results will be displayed in the calibration result interface. If the user confirms to use the calibration results, click "Write Calibration Results to Parameters". The system will then automatically adjust the robot's big and small arm lengths and the zero-position pulse of the second axis according to the calibration results.



2.4.5. Robot Zero Position Calibration

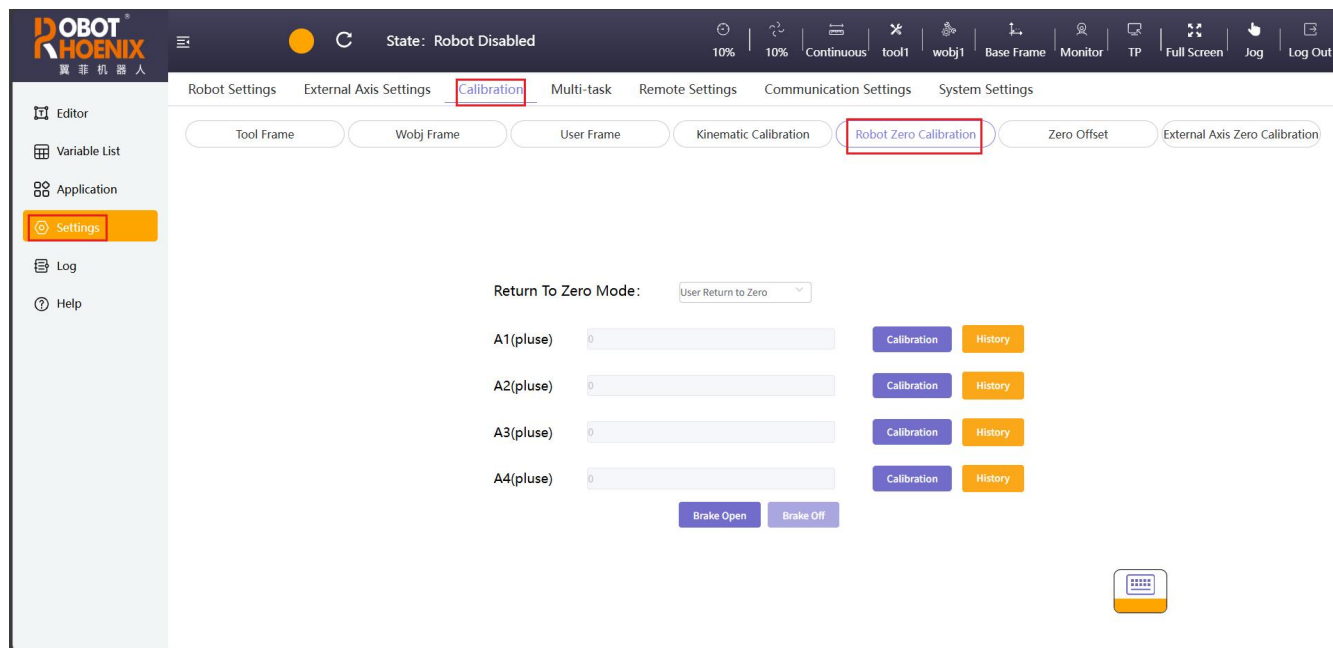
Gorilla Studio robot control system provides a robot zero position calibration function, which can perform single-axis zero position calibration.

The robot zero position calibration is to make the theoretical zero point in the control algorithm coincide with the actual mechanical zero point, so that the mechanical linkage system can correctly respond to the position and speed commands of the control system.

Before calibration, please confirm that the robot joint is at zero position, and do not calibrate at will.

Operation method

As shown in the figure below, click "Settings" - "Calibration" - "Robot Zero Position Calibration" in the left navigation bar to enter the robot zero position calibration interface for calibration. You can select the homing method (user homing or torque homing).



A1" - "A4" represent robot axes 1 to 4 respectively. Click the corresponding "Calibration" button to perform zero position calibration. Click the corresponding "History" to view the historical homing times, homing methods, and zero-position pulses.

Note:

Gorilla Studio robot control system supports manual opening and closing of the brake. Click the corresponding button to take effect.

When manually opening the brake, there is a risk of the robot falling. Please make sure it is safe before performing the operation.

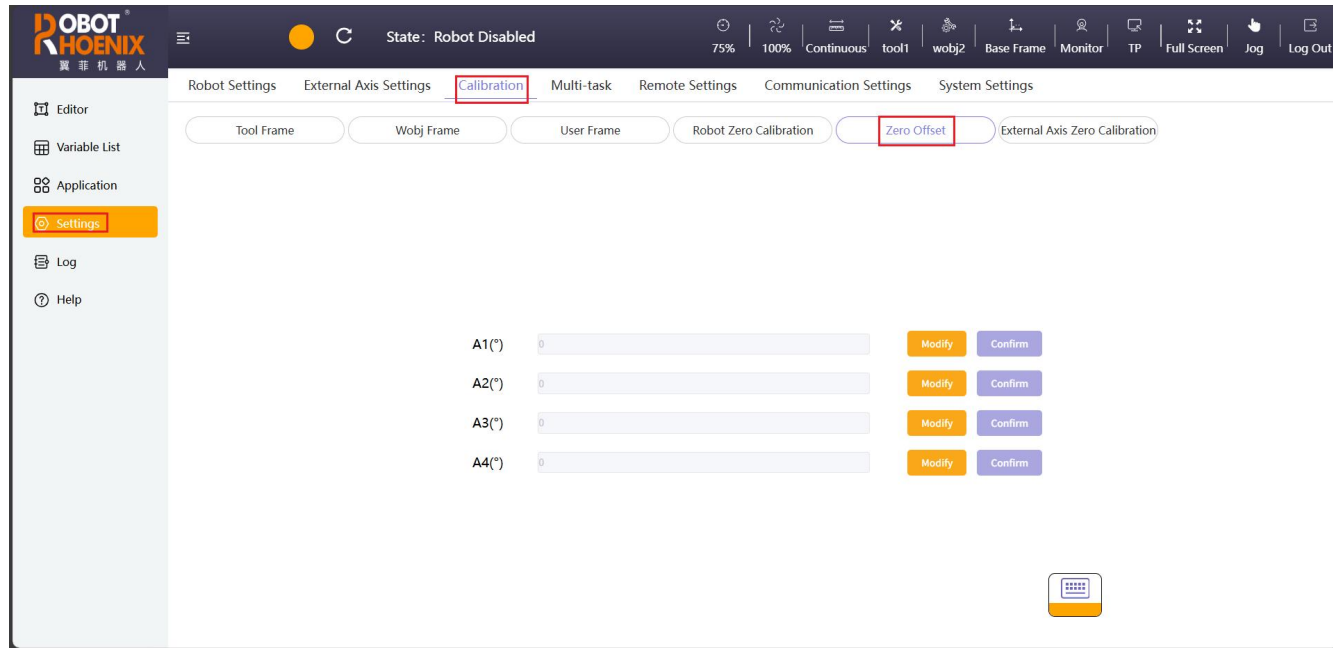
To prevent the need for re-teaching points due to skipped steps or loss of zero position, the fourth axis homing of SCARA robots can be performed using an L-shaped homing fixture for torque-based homing. Contact the manufacturer if needed. Instructions for using the homing fixture for torque-based homing can be found in the appendix.

2.4.6. Zero Offset

- Gorilla Studio robot control system provides a zero offset function.
- In some scenarios where space is limited, the robot cannot return to the mechanical zero point. If calibration is required at this time, the "zero offset" function needs to be used. This function can achieve the same effect as "robot zero calibration" by inputting a known angle when the current angle is known.
- When the actual zero position changes due to reasons such as belt toothing, the zero offset function can be used to correct it.

Operation method

As shown in the figure, click “Settings” → “Calibration” → “Zero Offset” in the left navigation bar to enter the Zero Offset interface for calibration.



"A1" - "A4" represent robot axes 1 to 4 respectively. Click the corresponding "Modify" button, enter the known angle value, and click "OK".

Note:

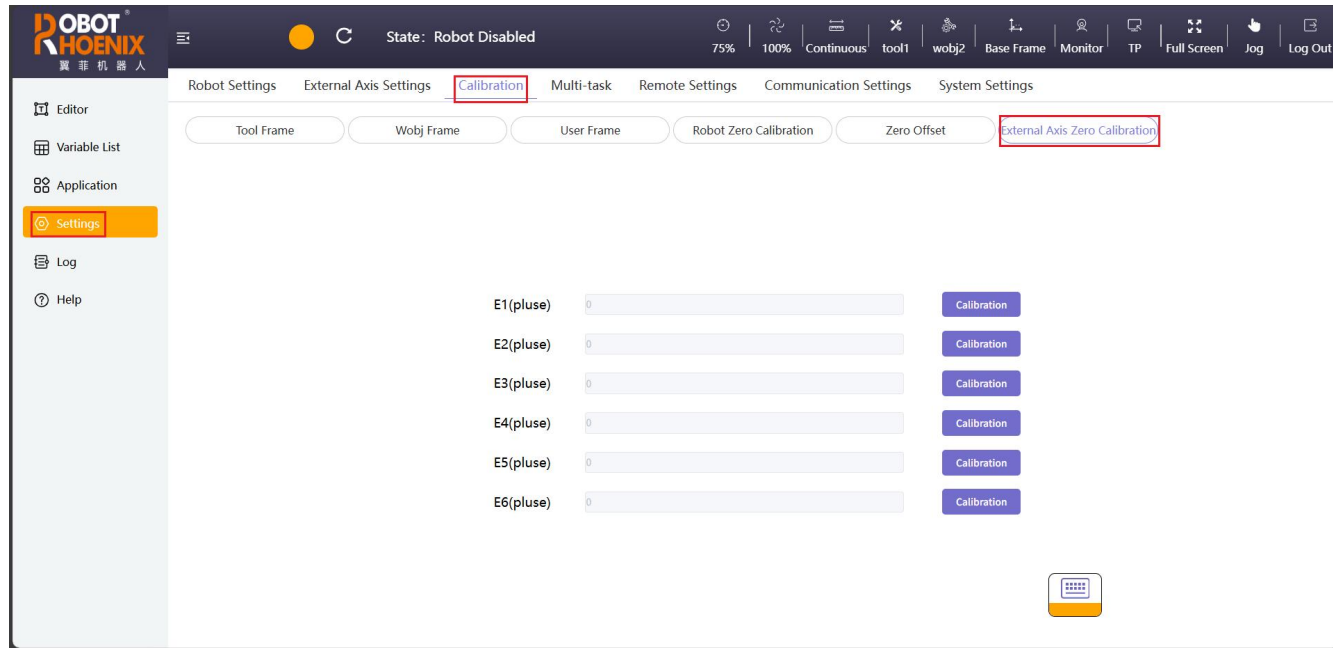
This function calibrates the zero position, rather than indicating that the zero position is the current angle.

2.4.7. External Axis Zero Position Calibration

Gorilla Studio robot control system provides an external axis zero position calibration function, which can perform single-axis zero position calibration.

Operation method

As shown in the figure below, click "Settings" - "Calibration" - "External Axis Zero Position Calibration" in the left navigation bar to enter the external axis zero position calibration interface for calibration.



"E1" - "E6" represent external axes 1 to 6 respectively. Click the corresponding "Calibration" button to perform zero position calibration.

2.5. Multi-tasking

2.5.1. Overview

The multitasking function can be used to execute multiple robot programs at the same time, supporting static and non-static startup. It is often used in the following situations:

- Continuously monitor a signal even if the main program is in single-step operation mode or the main program is paused in automatic operation mode. Similar to the background PLC function, but the response speed is not as fast as the real PLC.
- When the robot executes the main motion program, it sends or receives various information at the same time without being restricted by the execution logic of the main program.

Note:

Supports activating 4 multitasks at the same time, but not the same program.

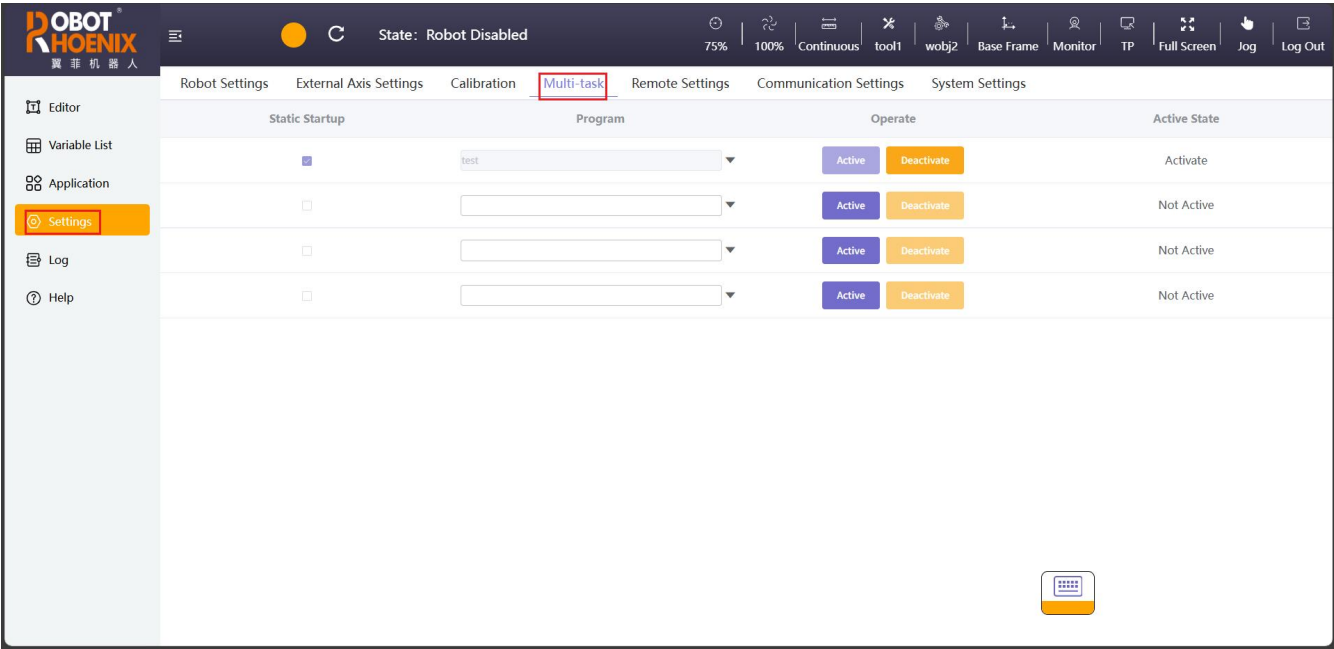
Motion commands are not supported in multitask programs.

2.5.2. Task Activation

Gorilla Studio robot control system provides a management interface for multiple tasks.

Operation method

As shown in the figure below, click "Settings" - "Multitasking" in the left navigation bar to enter the multitasking interface to manage multitasking programs.



Select the multi-task program to be activated from the drop-down box. If "Static Start" is checked, the program will start automatically when the robot is powered on. If "Static Start" is not checked, the program will not start immediately after power-on and will start or stop along with the main program. Click the "Activate" button to set the program status to "Activated". Click the "Deactivate" button to set the program status to "Inactive". At this point, you can re-select a multi-task program to be activated from the drop-down box.

Note:

If there are compilation errors in the multi-task program, an error will occur when activating it. You need to fix the errors and try again.

Once a multi-task program is activated, it cannot be edited or started.

2.5.3. Task Start and Stop

Operation method

- Start

Open the main program file to be run. After successful compilation, click "Start"/"Next" in the robot enabled state, or start the main program through a remote device, and the activated non-static multitask can be started and run synchronously.

- Stop

Open the main program file, click "Stop", or stop the main program through a remote device, and the activated non-static multitask can be stopped synchronously.

Note:

Normally, a multitasking will run in a loop. If a task does not contain any waiting commands, the background task may consume too many computer resources and cause the controller to be unable to process other tasks.

When a program is running, it is forbidden to modify the content in the multitasking management interface.

2.5.4. Inter-Task Communication

There are two ways to communicate between tasks: global variables and interrupts.

- Use global variables to communicate between tasks

Use global variables defined in the global variable table in all task projects that need to communicate.

Use global variables to control task execution, transfer data, etc. where needed.

- Use interrupts to coordinate execution order between tasks

Define interrupts and corresponding interrupt processing functions in the task that needs to be waited for.

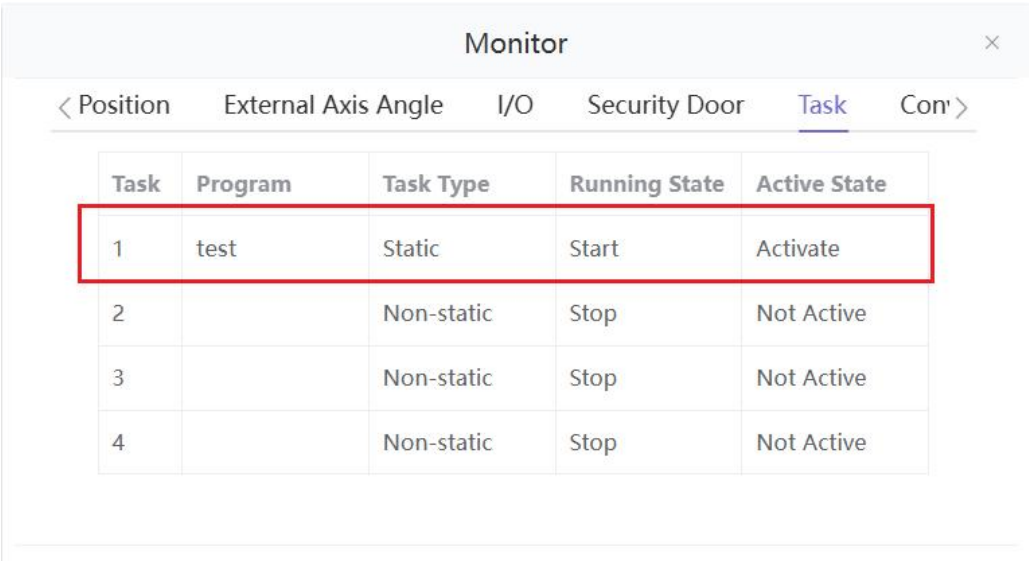
Set interrupt trigger signals in appropriate places of the task being waited for.

Note:

When using global variables and Wait or While commands to make one task wait for another task, it is necessary to pay attention to the waiting command (greater than 0.1s) to avoid the program from executing judgment statements too quickly and occupying too many system resources.

2.5.5. Task Monitoring

Click the "Monitor" button in the top toolbar to open the monitoring interface, and click "Task" to view the currently enabled tasks, as shown below:



Monitor				
< Position External Axis Angle I/O Security Door Task Com' >				
Task	Program	Task Type	Running State	Active State
1	test	Static	Start	Activate
2		Non-static	Stop	Not Active
3		Non-static	Stop	Not Active
4		Non-static	Stop	Not Active

Note:

To view local variables in a multitask program, switch the “Task Type” to the corresponding “Task.”

2.6. Remote Control

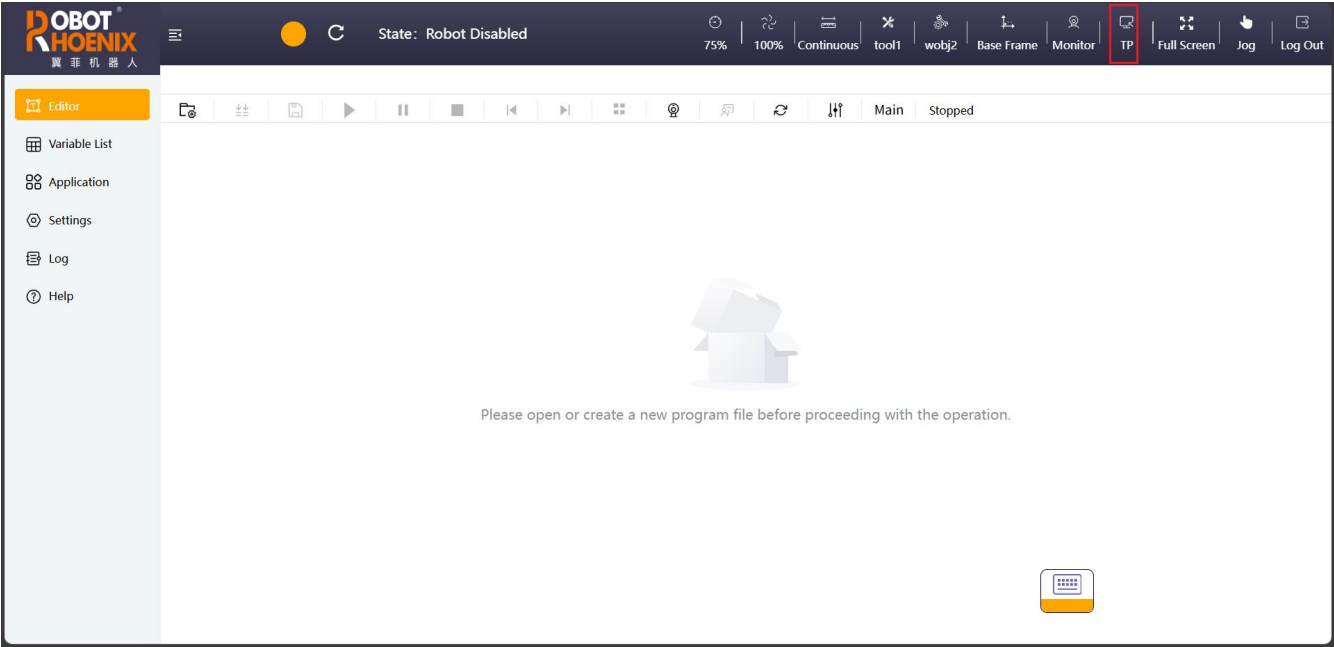
The Gorilla Studio robot control system supports four types of control devices: Teach Pendant, I/O, Modbus, and PLC.

The control device displayed when opening the Gorilla Studio robot control system is not necessarily the “Teach Pendant”; it depends on the previously configured control device. Switching control devices requires *Programmer* or higher-level permission and must be done when no program is running.

Control device switching can only be performed between the Teach Pendant and the other control devices, and not among the other control devices themselves. In other words, remote I/O control, remote Modbus control, and remote PLC control cannot be switched directly between each other.

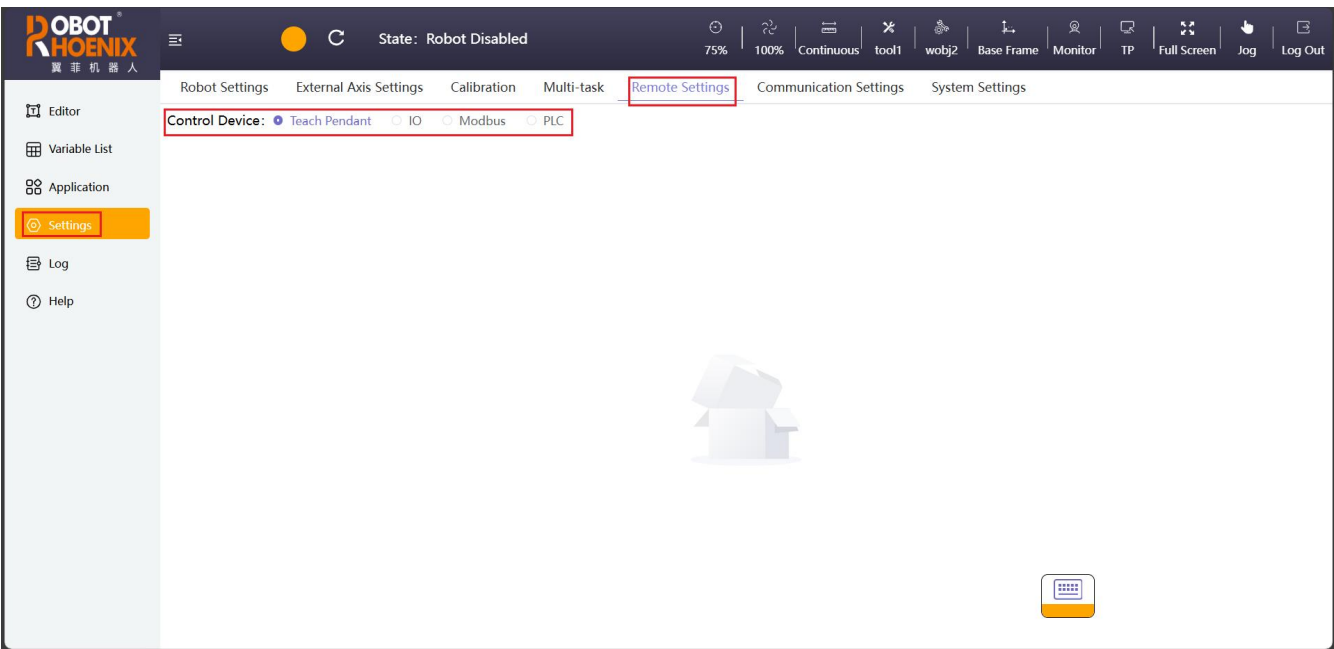
2.6.1. Teach Pendant Control

The system’s default control device at the factory is TP (Teach Pendant). As shown in the figure below:



To switch the Control Device, click the “” button and select the desired option from the dropdown menu. You can also switch it through the left navigation panel by going to “Settings” → “Remote Settings” and selecting the required control device.

The Teach Pendant can be switched to I/O, Modbus, or PLC.



Note:

When switching from remote control to Teach Pendant control, the previous output states will not be reset.

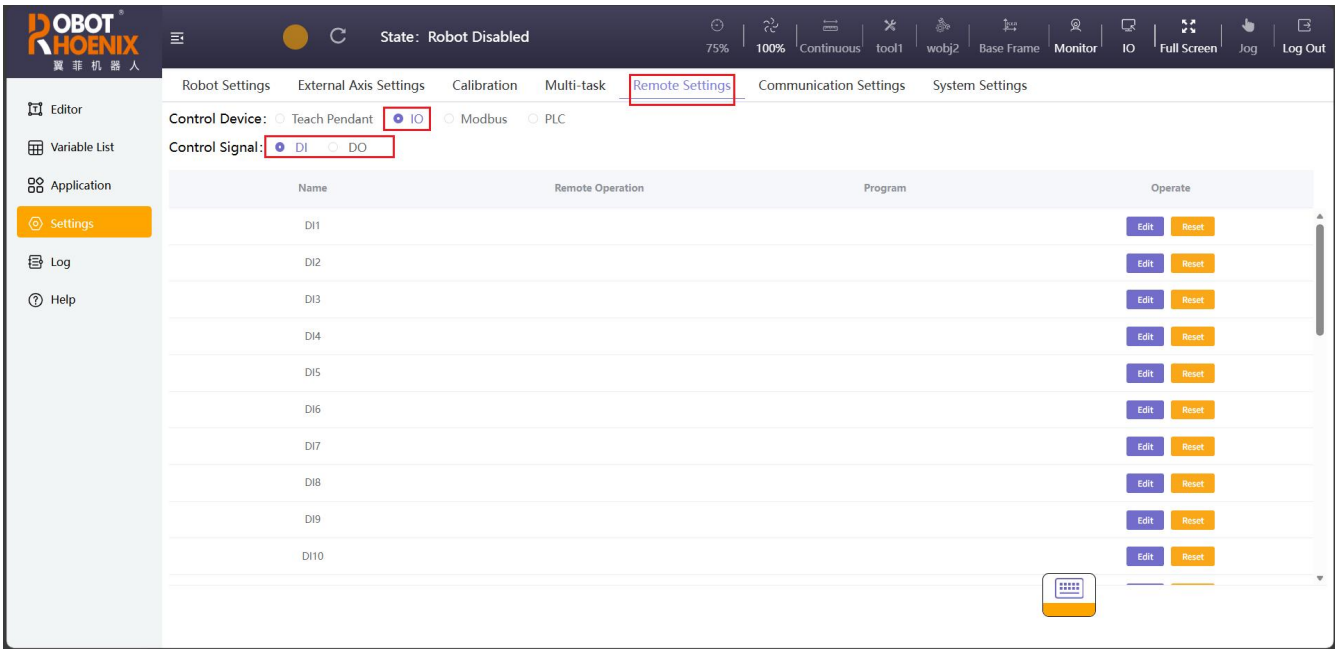
2.6.2. IO Control

Remote I/O control means that when the robot is in remote mode, program selection, program start, program pause, program reset, and program stop are executed through remote I/O signals, while the program running status is monitored through DO signals.

After selecting Remote I/O Control, both the web interface and the teach pendant only provide display and monitoring functions. For example, you can open and close program files, monitor robot positions and I/O status through the monitoring interface, perform variable monitoring, and display the current program line number, etc.

Operation Method

To switch the Control Device, click the “TP” button and select “I/O” from the dropdown menu. You can also switch it through the left navigation panel by going to “Settings” → “Remote Settings” and selecting “I/O” as the control device. As shown in the figure below:



Note:

Switching can only be done from the Teach Pendant to I/O, and from I/O back to the Teach Pendant. To switch to Modbus, you must first switch to the Teach Pendant, and then switch

from the Teach Pendant to Modbus.

Operation Method

After switching to I/O Control, follow the steps below:

Step 1: Configure DI Control Signals

Select the DI to be used for Program Selection, then click the corresponding “Edit” button. In the pop-up “Set Remote Control Function” window, select “Program Selection” from the dropdown menu. Next, choose the program file to be started from the “Select File” dropdown menu. After selection, click OK. Click the “Reset” button to clear the current DI operation settings.

Step 2: Configure DI Control Signals

Select the DI to be used for Start, then click the corresponding “Edit” button. In the pop-up window, select “Start” from the dropdown menu and click OK. Click “Reset” to clear the current DI operation settings.

Step 3: Configure DI Control Signals

Select the DI to be used for Stop, click “Edit”, select “Stop” from the dropdown menu, and click OK. Click “Reset” to clear the current DI operation settings.

Step 4: Configure DI Control Signals

Select the DI to be used for Pause, click “Edit”, select “Pause” from the dropdown menu, and click OK. Click “Reset” to clear the current DI operation settings.

Step 5: Configure DI Control Signals

Select the DI to be used for Reset, click “Edit”, select “Reset” from the dropdown menu, and click OK. Click “Reset” to clear the current DI operation settings.

Step 6: Configure DO Control Signals

Select the DO corresponding to Program Running, click “Edit”, select “Program Running” from the dropdown menu, and click OK. Click “Reset” to clear the current DO operation settings.

Step 7: Configure DO Control Signals

Select the DO corresponding to Program Paused, click “Edit”, select “Program Paused”, and click OK. Click “Reset” to clear the current DO operation settings.

Step 8: Configure DO Control Signals

Select the DO corresponding to Program Error, click “Edit”, select “Program Error”, and click

OK. Click “Reset” to clear the current DO operation settings.

Step 9: Configure DO Control Signals

Select the DO corresponding to Emergency Stop Status, click “Edit”, select “Emergency Stop Status”, and click OK. Click “Reset” to clear the current DO operation settings.

Step 10: Activate the corresponding DIs to remotely control the robot. By monitoring the corresponding DO signal status, you can track the program’s running status.

Note:

Multiple Program Selection operations can be configured, meaning multiple program files can be controlled using DI signals. However, the other operations (Start, Pause, Stop, Reset) can only be configured with a single DI.

If multiple Program Selection operations are configured, the first program will be executed according to the DI order. For example, if DI1 corresponds to Program 1, DI2 to Program 2, DI3 to Program 3, and DI4 to Start, and DI1, DI2, DI3, and DI4 are all set to True, the operation result will be that Program 1 is started.

For DO control signals corresponding to operations (Program Running, Program Paused, Program Error, Emergency Stop Status), each operation can only be configured with a single DO.

2.6.3. Modbus Control

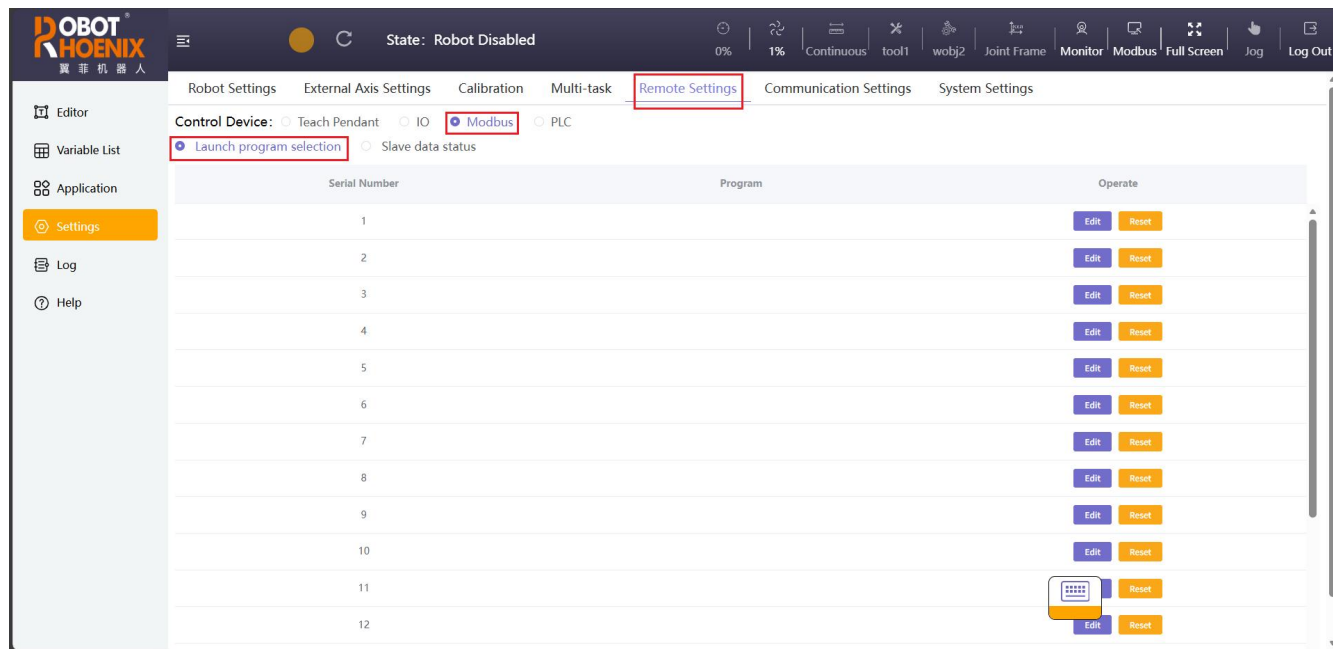
Modbus Control refers to using the remote Modbus control function to control program start, stop, and other operations by writing to the corresponding addresses from the master station.

Note:

Before using Modbus control, “2.7.3 Modbus Configuration” must be completed.

Operation Method

To switch the Control Device, click the  button and select “Modbus” from the dropdown menu. You can also switch it through the left navigation panel by going to “Settings” → “Remote Settings” and selecting “Modbus” as the control device. As shown in the figure below:

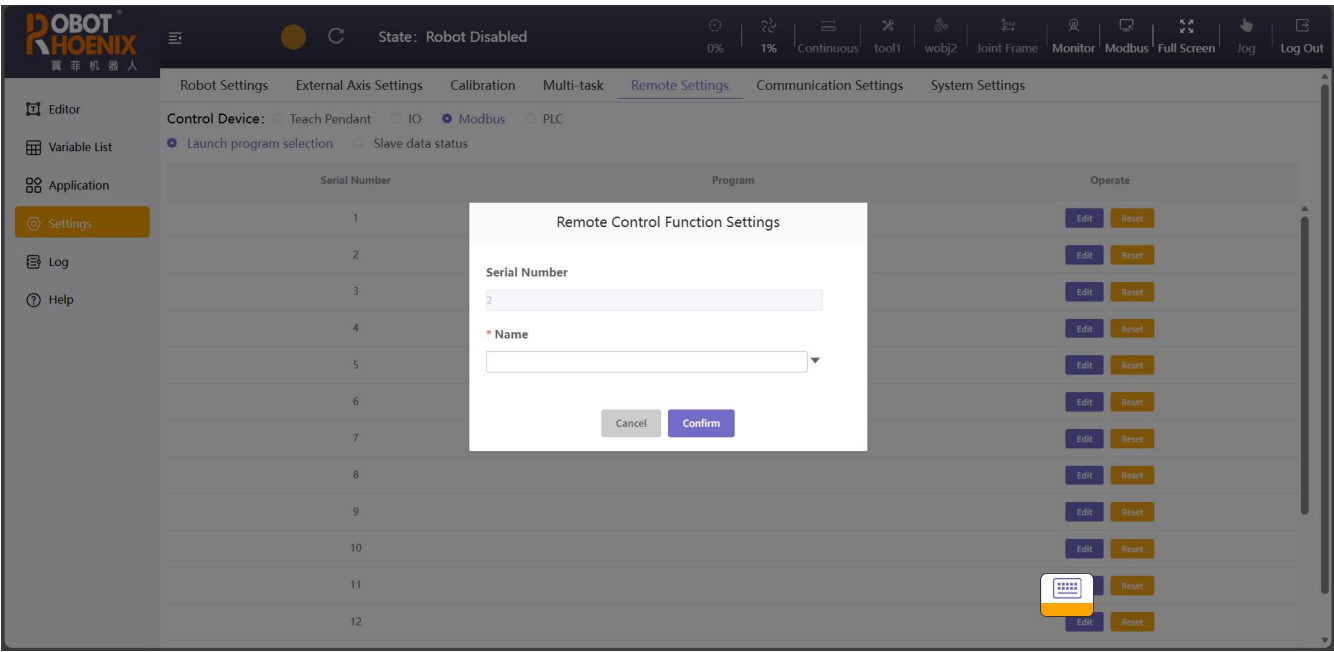
**Note:**

Switching can only be done from the Teach Pendant to Modbus, and from Modbus back to the Teach Pendant. To switch to I/O, you must first switch to the Teach Pendant, and then switch from the Teach Pendant to I/O.

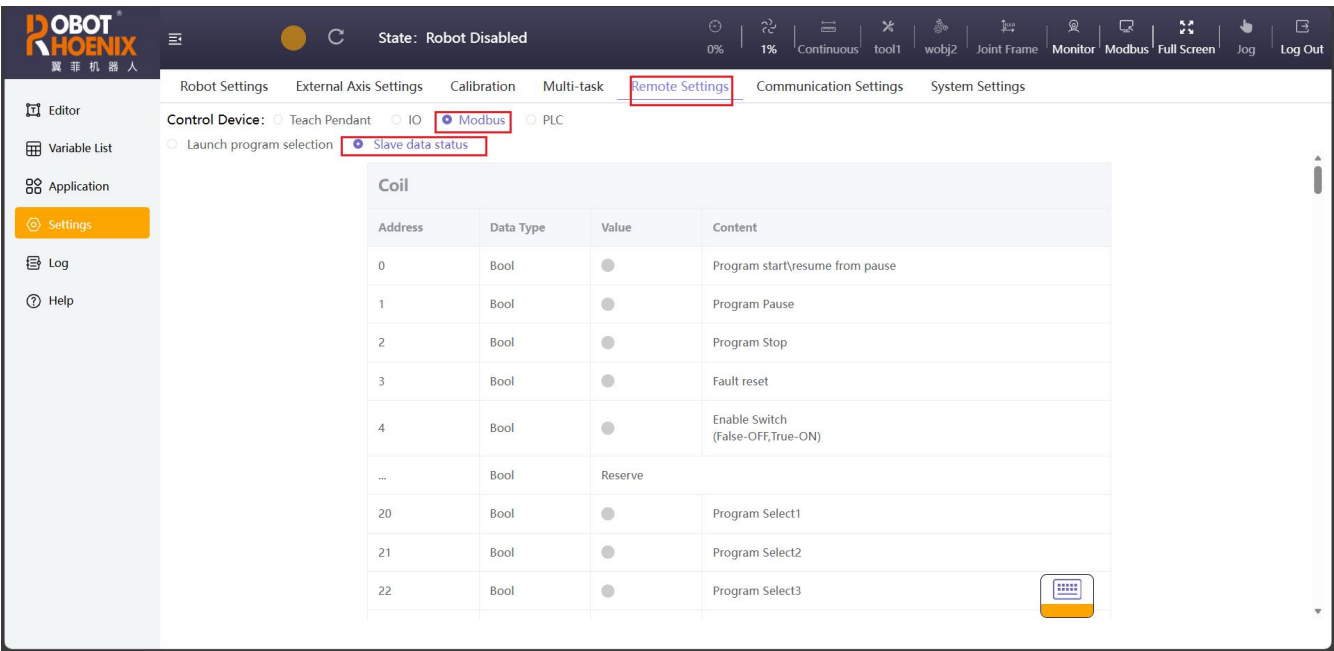
After switching to Modbus Control, follow the steps below:

Step 1: Select the Program to Start

Click the “Edit” button corresponding to the Modbus number. In the pop-up “Set Remote Control Function” window, select the program file to be started from the “Program Name” dropdown menu. After selection, click OK. Click the “Reset” button to clear the currently selected program files. Up to 20 programs can be selected for startup.



Step 2: Click “Slave Data Status” to view the status of the slave station, as shown in the figure below:

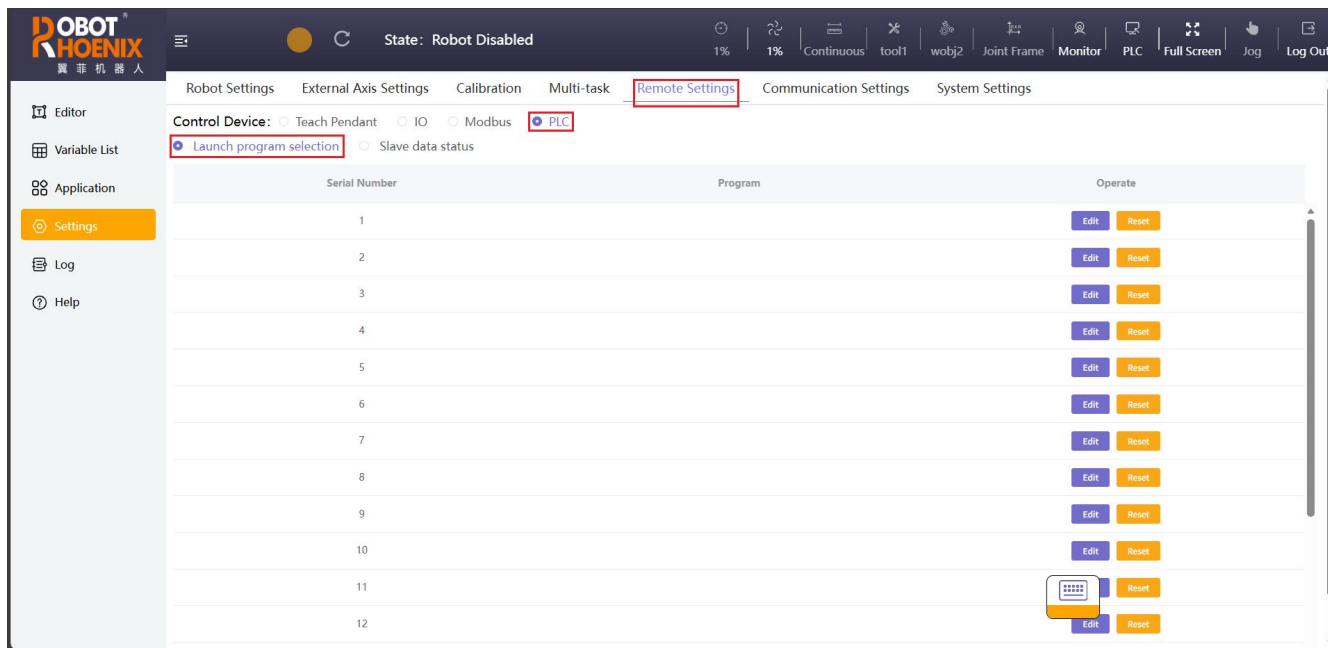


2.6.4. PLC Control

When the PLC establishes a communication connection with the robot via Profinet or EtherNet/IP bus, the robot can be operated through the assigned remote control addresses. This function uses the same program startup list as the Modbus remote control method.

Operation Method

To switch the Control Device, click the “TP” button and select “PLC” from the dropdown menu. You can also switch it through the left navigation panel by going to “Settings” → “Remote Settings” and selecting “PLC” as the control device. As shown in the figure below:



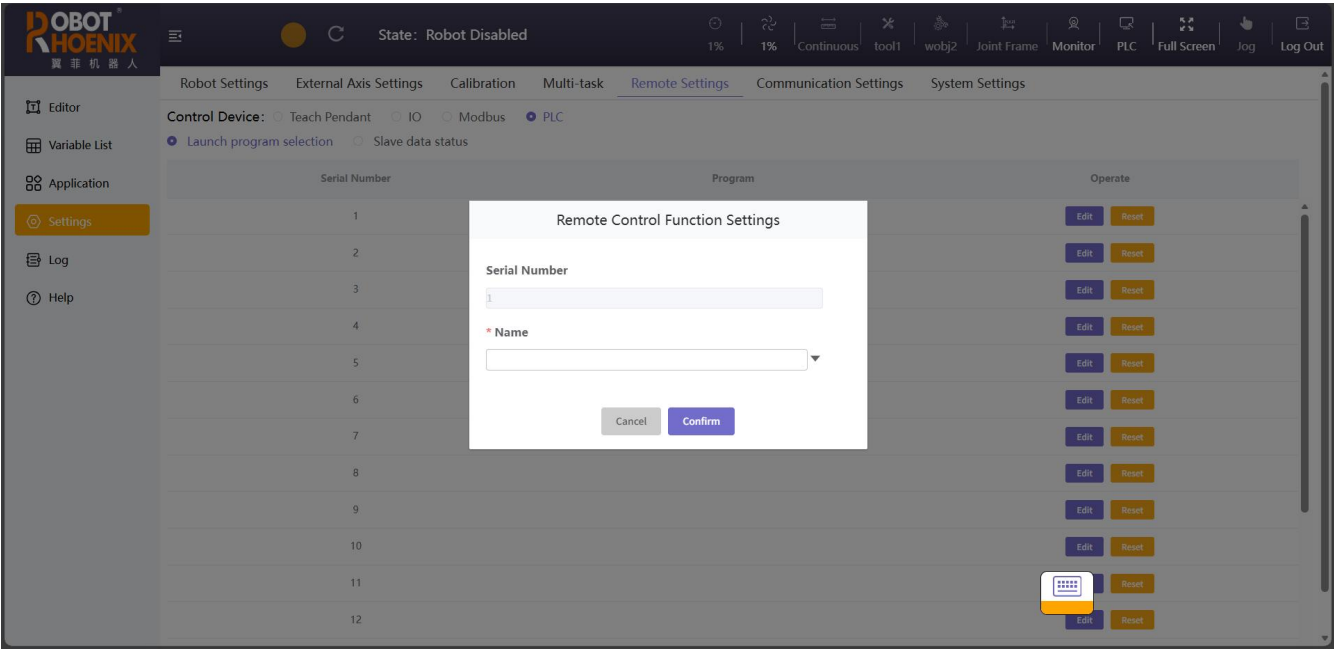
Note:

Switching can only be done from the Teach Pendant to PLC, and from PLC back to the Teach Pendant. To switch to other devices, you must first switch to the Teach Pendant, and then switch from the Teach Pendant to the desired device.

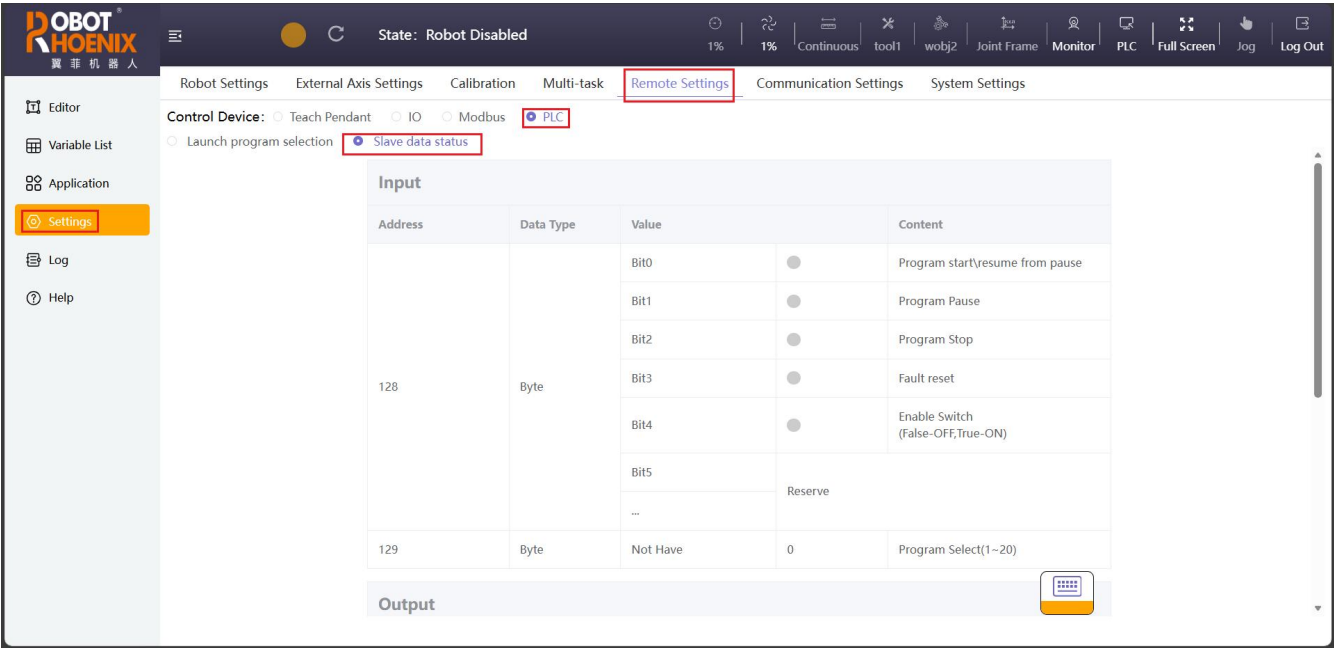
After switching to PLC Control, follow the steps below:

Step 1: Select the Program to Start

Click the “Edit” button corresponding to the PLC number. In the pop-up “Set Remote Control Function” window, select the program file to be started from the “Program Name” dropdown menu. After selection, click OK. Click the “Reset” button to clear the currently selected program files. Up to 20 programs can be selected for startup.



Step 2: Click “Slave Data Status” to view the status of the slave station, as shown in the figure below:

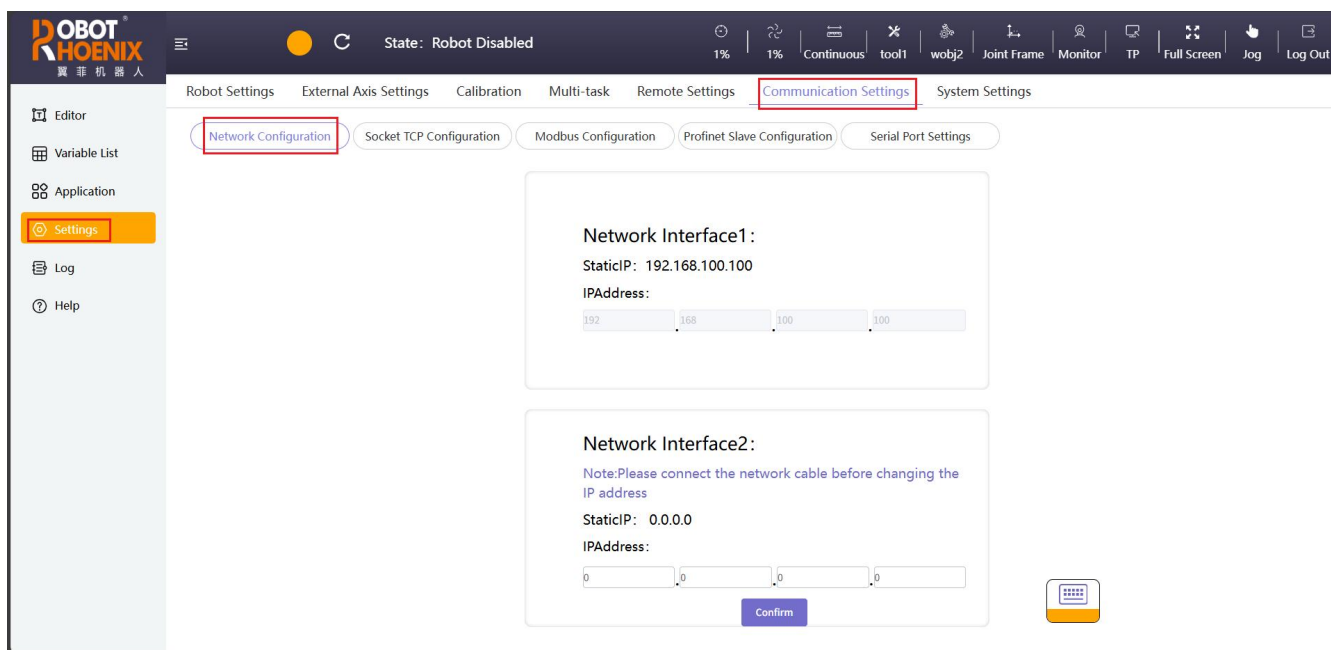


2.7. Communication Settings

The Gorilla Studio robot control system supports Socket TCP communication and Modbus communication.

2.7.1. Network Configuration

The network configuration includes Ethernet Port 1 and Ethernet Port 2. One port must be connected to the controller first, while the other port can be connected according to the user's needs, as shown in the figure below:



2.7.2. SocketTCP Configuration

To communicate with external devices, the robot can be configured as either a Client or a Server.

This function is commonly used for vision systems, where the robot is specified as a Client or Server in the vision function. The hardware has Ethernet Port 1 and Ethernet Port 2, so when the robot is configured as either a Client or Server, both ports support communication with third-party devices.

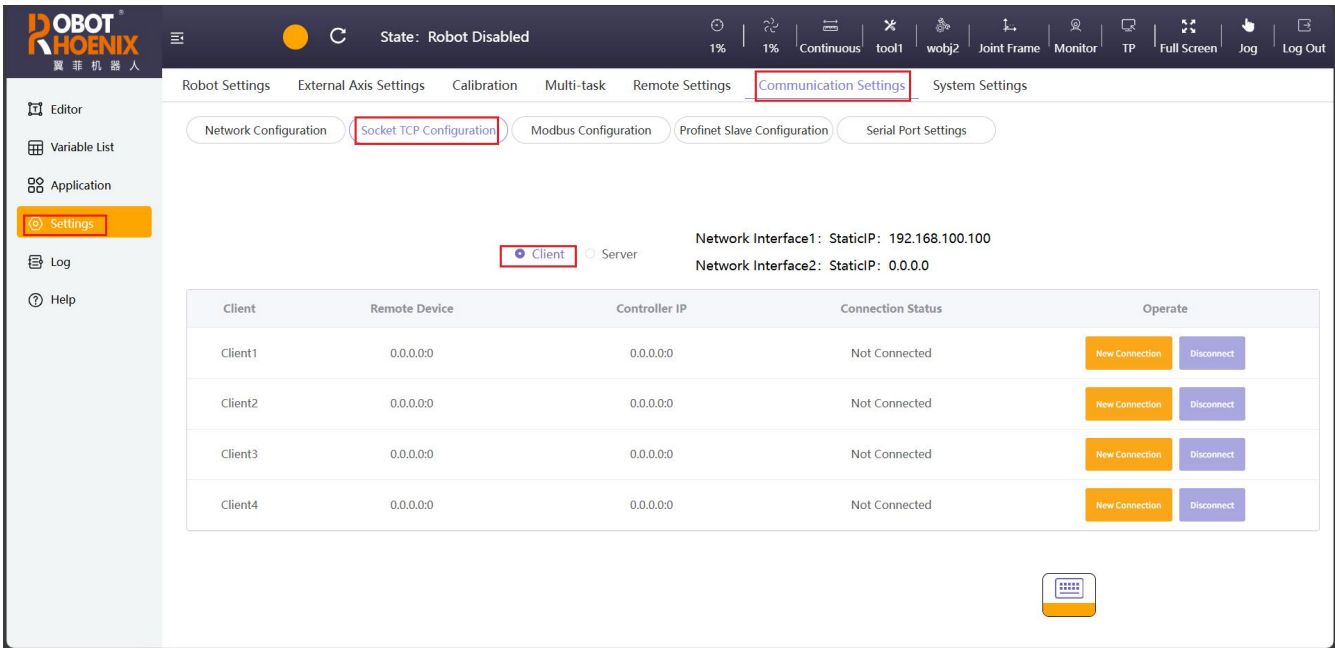
2.7.2.1. Client Mode

2.7.2.1.1. Create New Connection

When the controller is configured as a Client, it connects to other server devices to perform TCP communication.

Operation Method

Click the “Create New Connection” button for the corresponding client, enter the server’s IP address and port number, and then click OK. As shown in the figure below:



Once the connection is successful, the Connection Status will change from “Disconnected” to “Connected”.

Note:

The controller can establish up to four connections when configured as a Client.

In Client mode, each port number can only connect to a single server.

The valid port number range is 1025–49151.

2.7.2.1.2. Disconnect

When the controller is configured as a Client and does not want to connect to a specific server, this operation can be performed to disconnect from the specified server.

Operation Method

Click the corresponding “Disconnect” button in the list.

After disconnecting, the Connection Status will change from “Connected” to “Disconnected”, and the disconnected server will no longer be able to communicate with the controller.

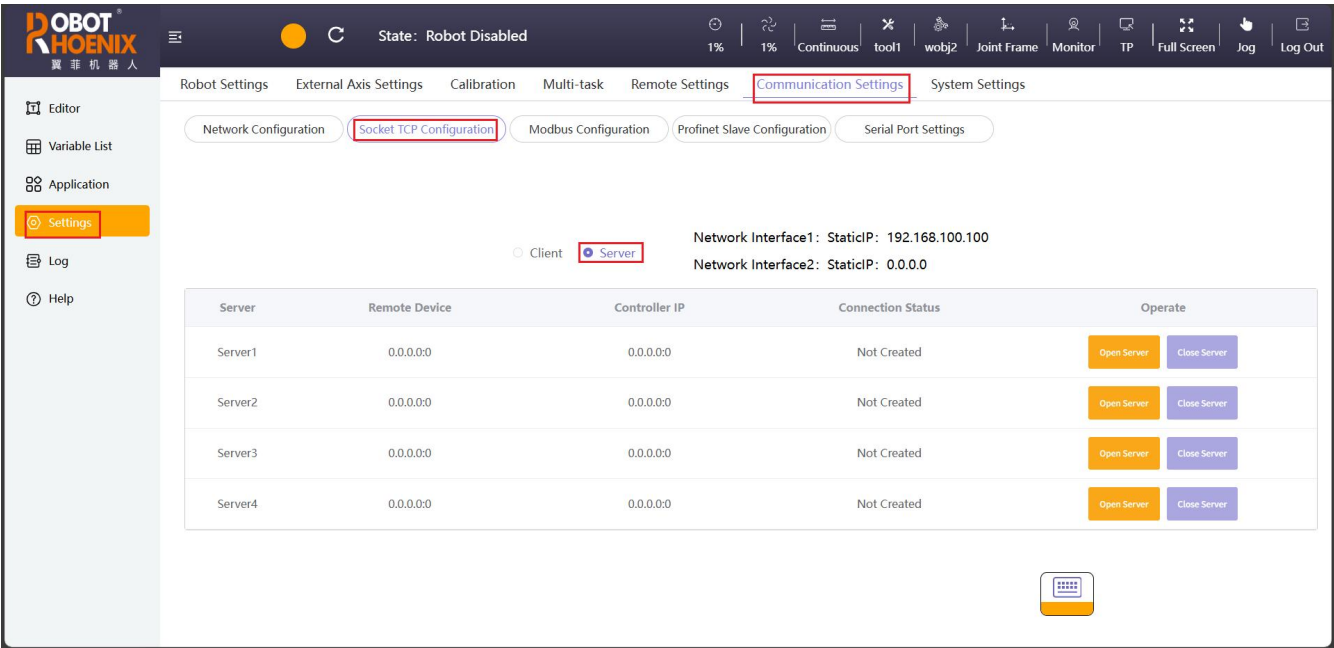
2.7.2.2. Server Mode

2.7.2.2.1. Start Server

When the server is in the “Not Created” state, performing the “Start Server” operation on the controller will configure it as a Server and set it to the active state. Other devices configured as clients (e.g., vision software) can directly connect to the controller for communication.

Operation Method

Click the “Start Server” button for the corresponding server, enter the controller’s IP address and port number, and then click OK. As shown in the figure below:



Note:

As a Server, up to four client connections are supported.

If the server has not been created or creation fails, the Connection Status for the corresponding entry will display “Not Created”.

If the server is created successfully but no clients are connected, the Connection Status will display “Created”.

If the server is created successfully and a client connection is established, the Connection Status will display “Connected”.

The valid port number range is 1025–49151.

2.7.2.2.2. Shutdown Server

Shut down the current controller in Server Mode to prevent other clients from connecting and communicating with the controller.

Operation Method

Click the corresponding “Shutdown Server” button.

After disconnecting, the Connection Status will change from “Created” or “Connected” to “Not Created”.

2.7.3. Modbus Configuration

The Gorilla Studio robot control system supports Modbus communication. It supports both TCP and RTU protocols, as well as Master and Slave bus configurations.

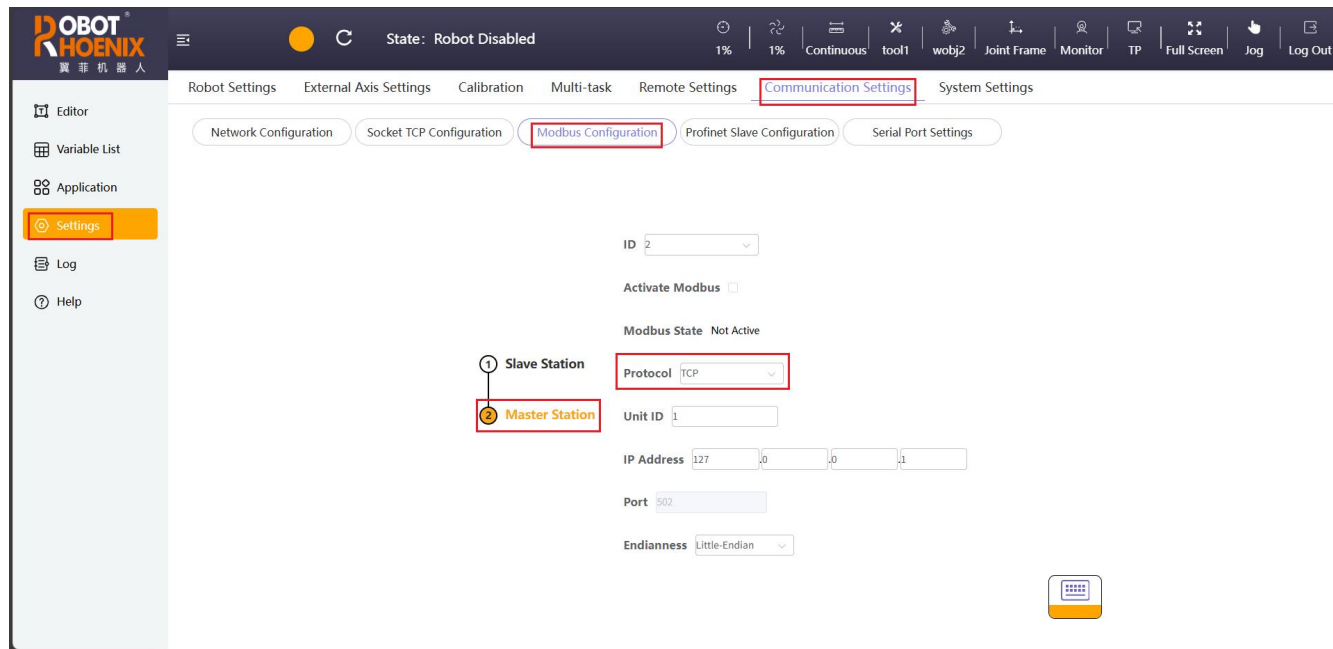
2.7.3.1. Modbus TCP Configuration

2.7.3.1.1. The Master

Select the robot as the Master.

Operation Method

Click “Settings” → “Communication Settings” → “Modbus Configuration” → “Master”, select “TCP” for the Protocol, enter the IP address and station number of the slave to communicate with, choose the byte order, and then check “Activate Modbus”. As shown in the figure below:

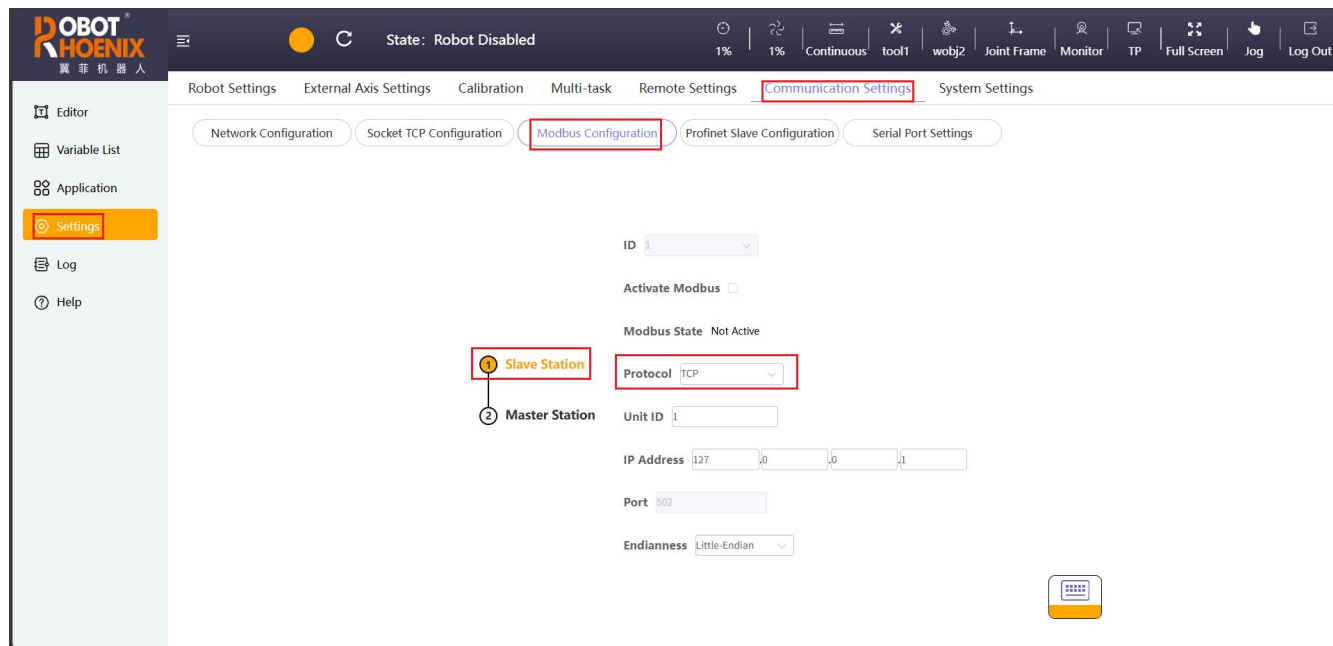


2.7.3.1.2. The Slave

Select the robot as the Slave.

Operation Method

Click “Settings” → “Communication Settings” → “Modbus Configuration” → “Slave”, select “TCP” for the Protocol, enter the IP address and station number, choose the byte order, and then check “Activate Modbus”. As shown in the figure below:



Note:

After successful activation, the Modbus Status will change from “Not Activated” to “Activated”, and the station number, IP address, and byte order will be read-only. When the Modbus Status is “Activated”, unchecking “Activate Modbus” will deactivate it, allowing the station number, IP address, and byte order to be modified.

The valid station number range is 1–255.

The port number is fixed at 502.

The byte order options include Little Endian, Big Endian, Little Endian with byte swap, and Big Endian with byte swap.

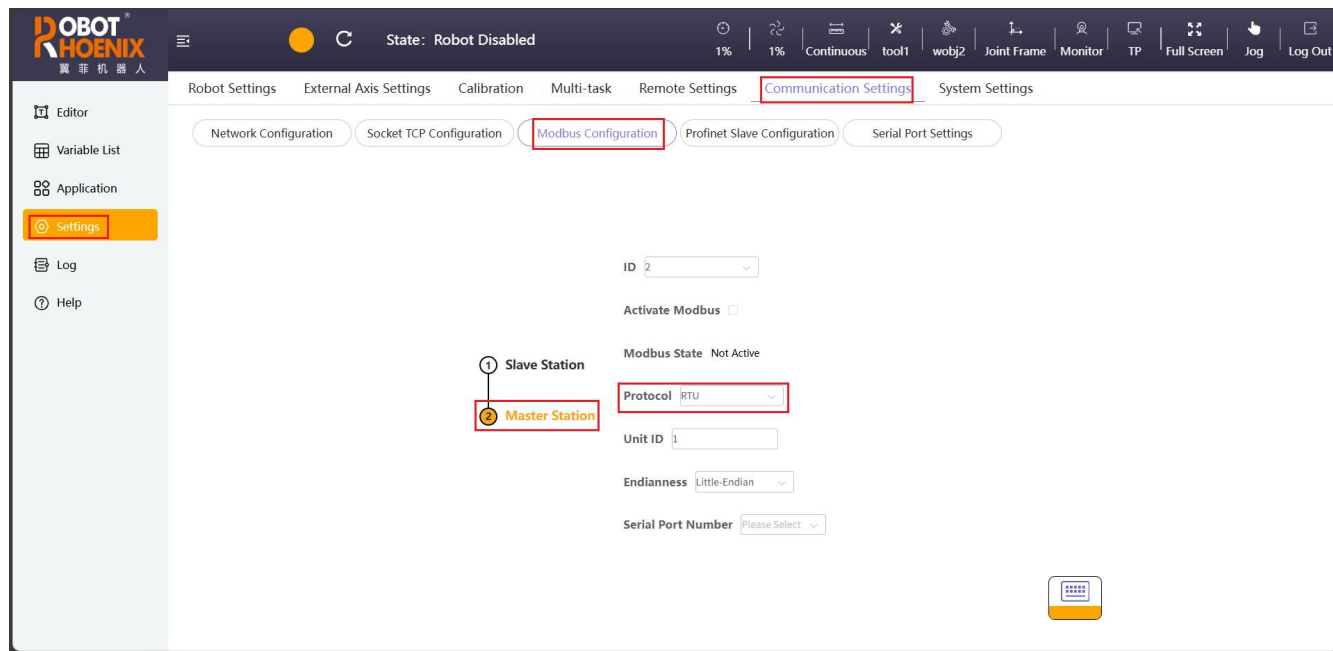
2.7.3.2. Modbus RTU Configuration

2.7.3.2.1. The Master

Select the robot as the Master.

Operation Method

Click Settings → Communication Settings → Modbus Configuration → The Master, select Protocol as RTU, enter the Station ID, choose the Byte Order, and select the Serial Port (this must be configured in Settings → Communication Settings → Serial Port Settings). Check Activate Modbus to enable it, as shown in the figure below:

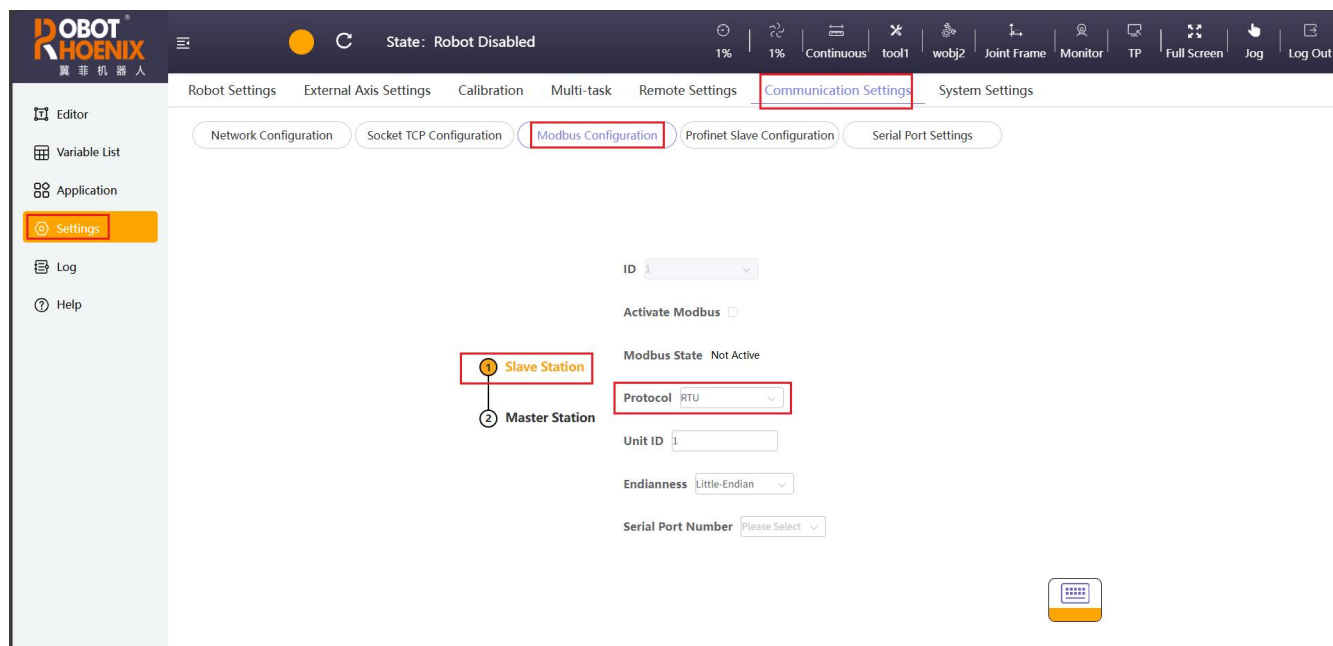


2.7.3.2.2. The Slave

Select the robot as the Slave.

Operation Method

Click “Settings” → “Communication Settings” → “Modbus Configuration” → “The Slave”, select “RTU” for the Protocol, enter the station number, choose the byte order, and select the serial port (this must be configured under “Settings” → “Communication Settings” → “Serial Port Settings”). Then check “Activate Modbus”. As shown in the figure below:



Note:

After successful activation, the Modbus Status will change from “Not Activated” to “Activated”, and the station number, byte order, and serial port will become read-only. When the Modbus Status is “Activated”, unchecking “Activate Modbus” will deactivate it, allowing the station number, byte order, and serial port to be modified.

The valid station number range is 1–255.

The byte order options include Little Endian, Big Endian, Little Endian with byte swap, and Big Endian with byte swap.

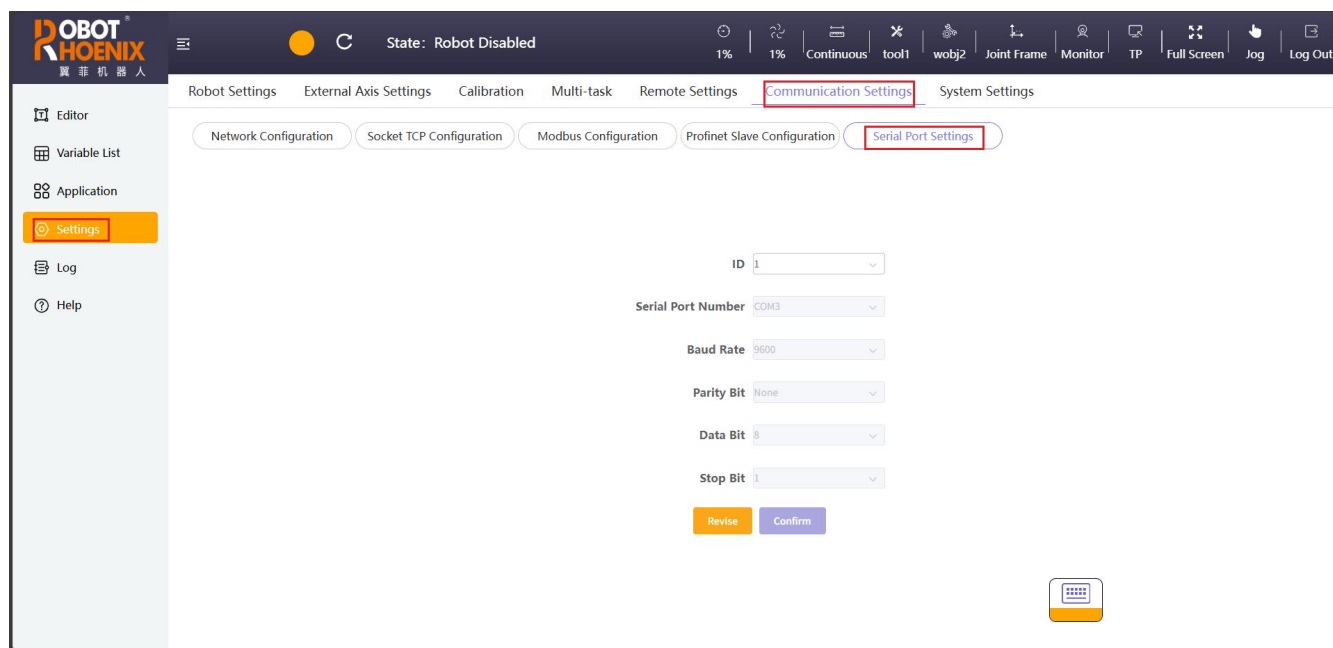
2.7.4. Serial Port Settings

The Gorilla Studio robot control system supports communication with external devices via serial ports. Using serial ports requires hardware support. Serial communication can be performed through the RS-485 port on the robot control cabinet, or via the USB port on the control cabinet using a USB-to-RS-485 converter module.

Operation Method

Click “Settings” → “Communication Settings” → “Serial Port Settings” to enter the serial port settings interface.

Select the serial port number, choose “COM3” for the port, set the baud rate, parity, data bits, and stop bits, and then click OK. As shown in the figure below:



Note:

Baud Rate: Options include 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

Parity: Options include Odd, Even, Mark (all 1s), Space (all 0s), and None.

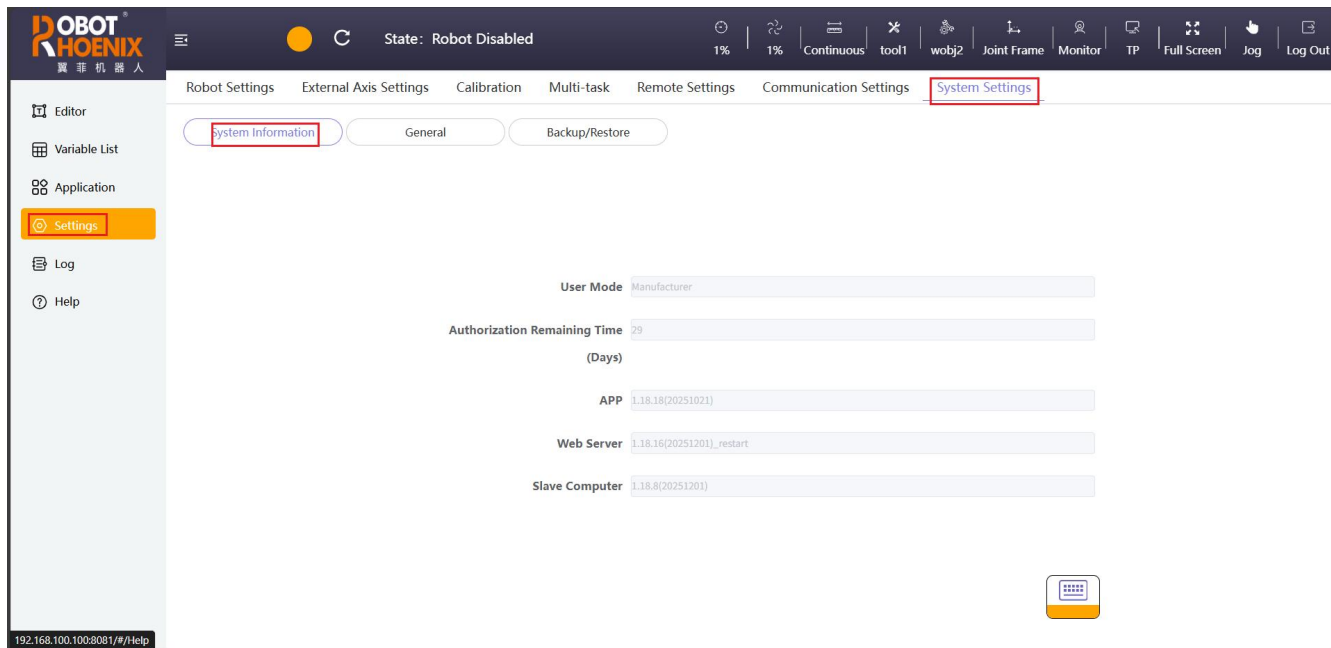
Data Bits: Options include 5, 6, 7, 8.

Stop Bits: Options include 1, 2.

2.8. System Settings

2.8.1. System Information

The **System Information** includes the currently logged-in username, remaining license time, and system version information.



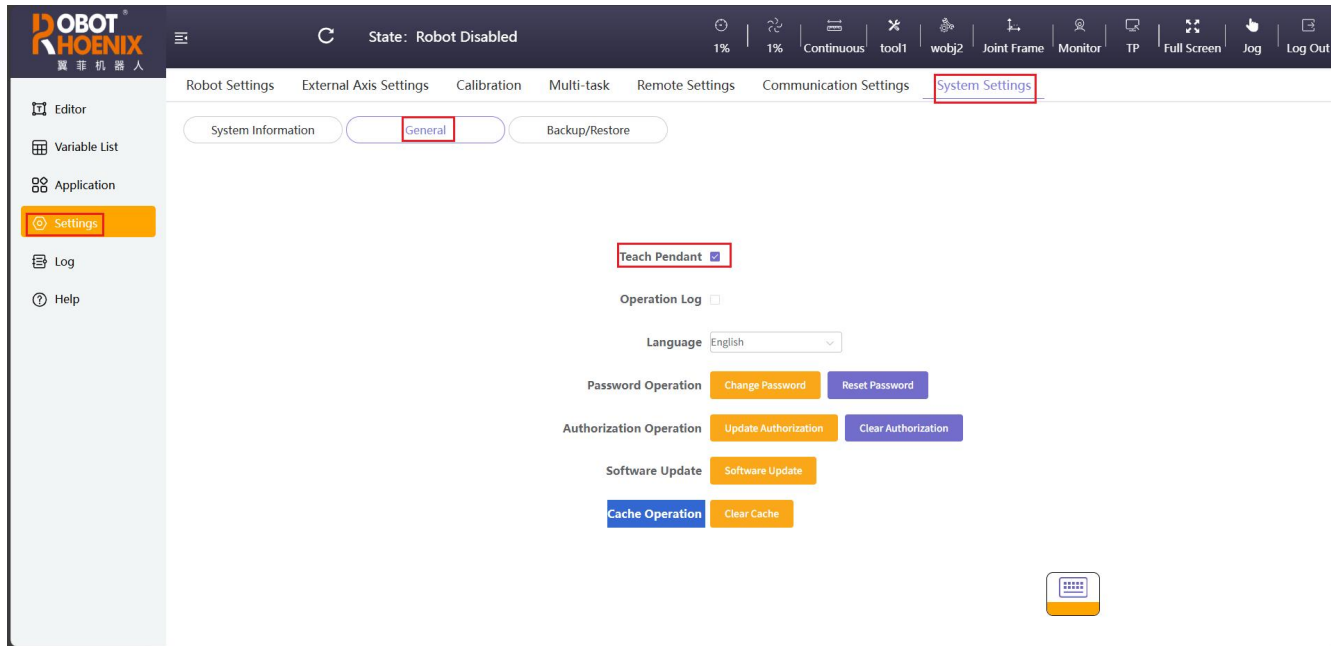
2.8.2. General

You can configure the Teach Pendant mode, program editing type, and system language. Additionally, you can change or reset passwords, update or clear licenses, perform software updates, and clear the cache.

2.8.2.1. Teach Pendant Mode

Operation Method

Check the “Teach Pendant” button to switch to Teach Pendant Mode, as shown in the figure below:

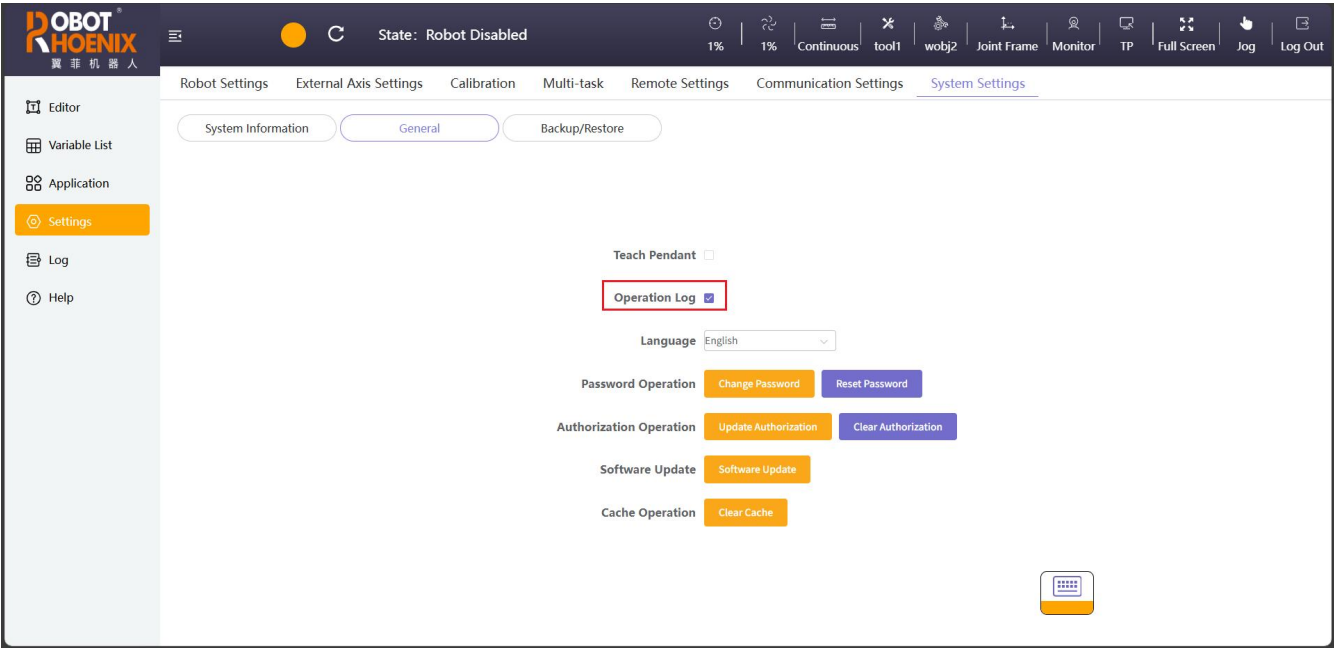


2.8.2.2. Operation Log

When this feature is enabled, all operations performed by the user through the host computer will be fully recorded in the log file.

Operation Method

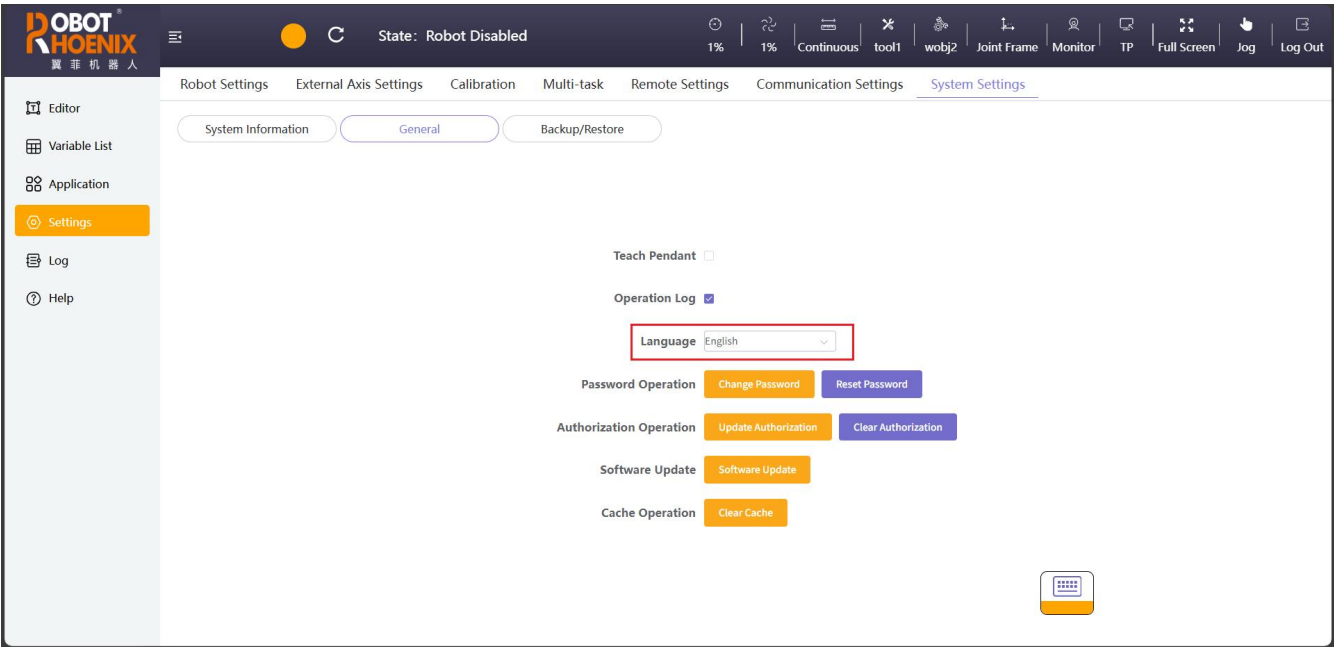
Click “Settings” → “System Settings” → “General” and check “Operation Log” to enable the logging feature. As shown in the figure below:



2.8.2.3. Language

Operation Method

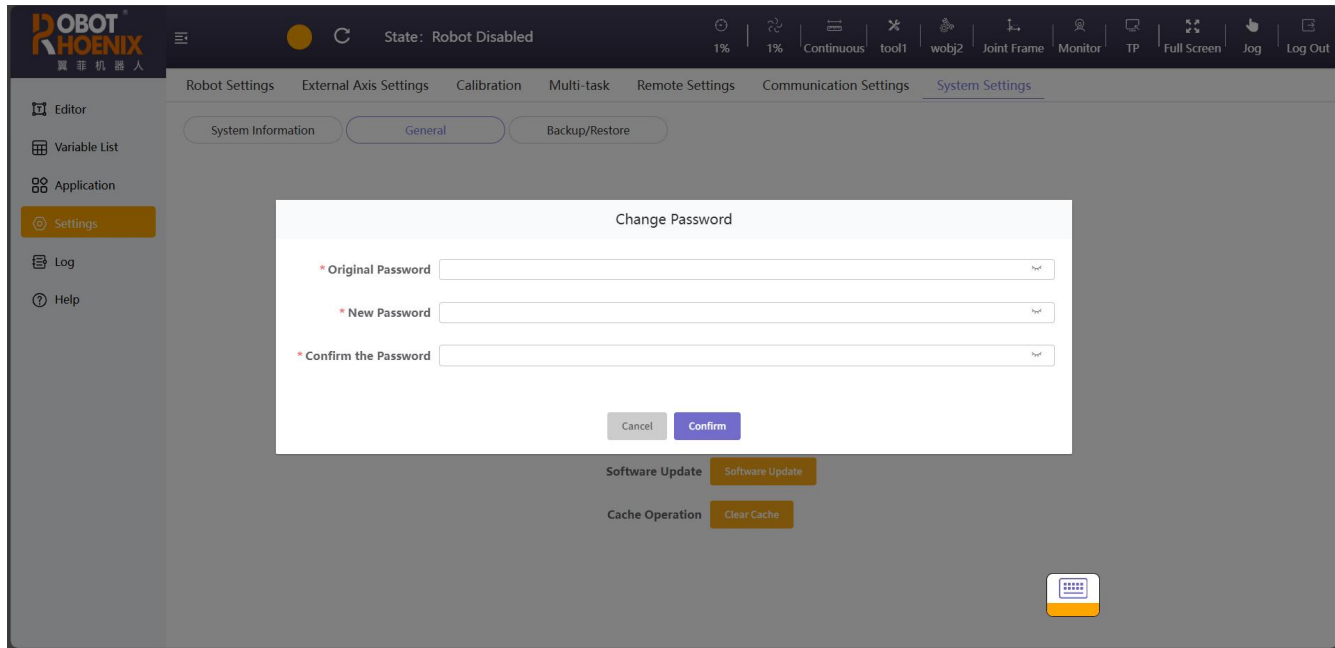
Click the “Language” dropdown menu to select the system language. Chinese and English are supported. As shown in the figure below:



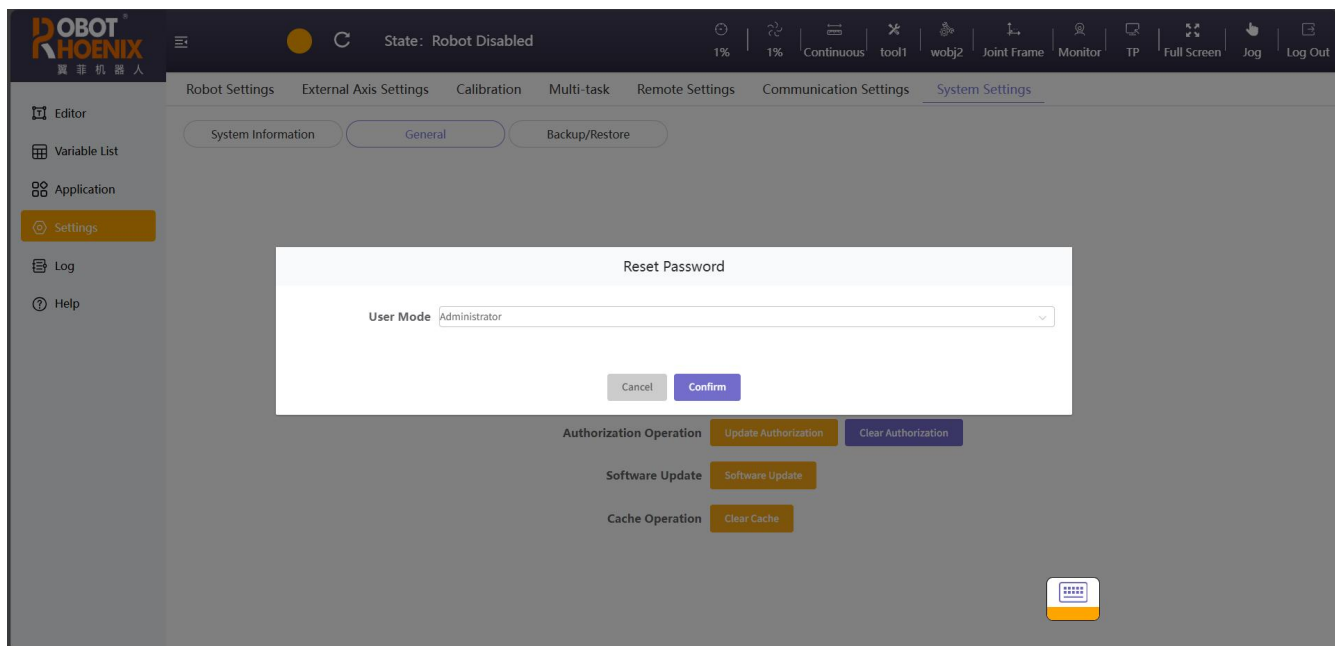
2.8.2.4. Change and Reset Password

Operation Method

To change the password, click the “Change Password” button. In the pop-up Change Password interface, make the desired modifications and then click OK. As shown in the figure below:



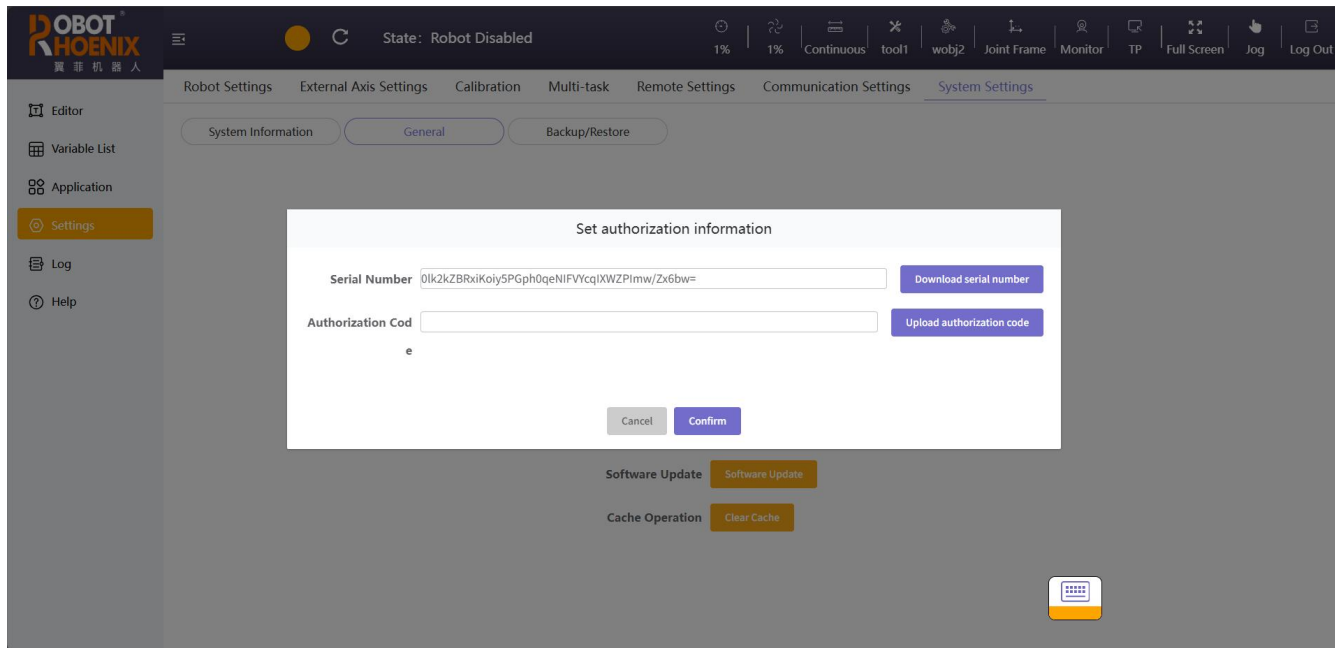
To reset a password, switch to “Manufacturer” permission, then click the “Reset Password” button. In the pop-up Reset Password interface, click the dropdown menu, select the user whose password needs to be reset, and then click OK. As shown in the figure below:



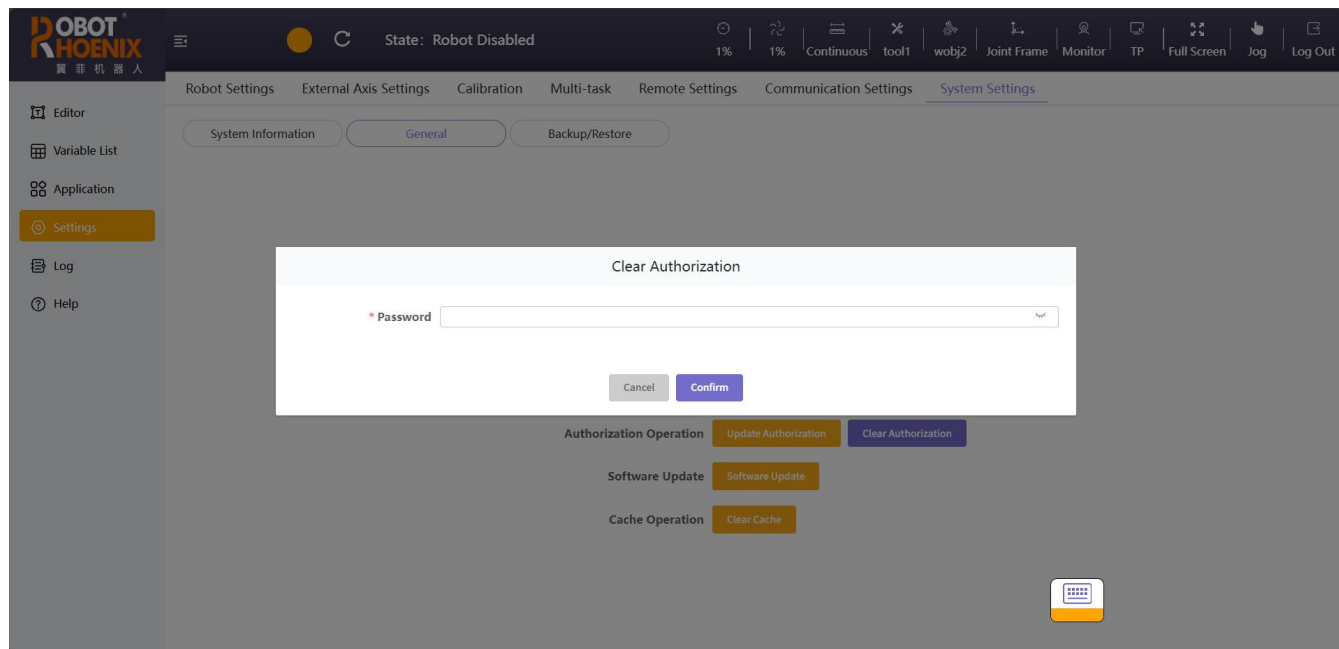
2.8.2.5. Update License and Clear License

Operation Method

To update the license, click “Settings” → “System Settings” → “General” → “Update License”. In the pop-up Set License Information interface, enter or upload the license code (send the serial number to the manufacturer to obtain the license code), and then click OK. As shown in the figure below:



To clear the license, switch to “Manufacturer” permission, then click “Settings” → “System Settings” → “General” → “Clear License”. In the pop-up Clear License interface, enter the password and click OK. As shown in the figure below:

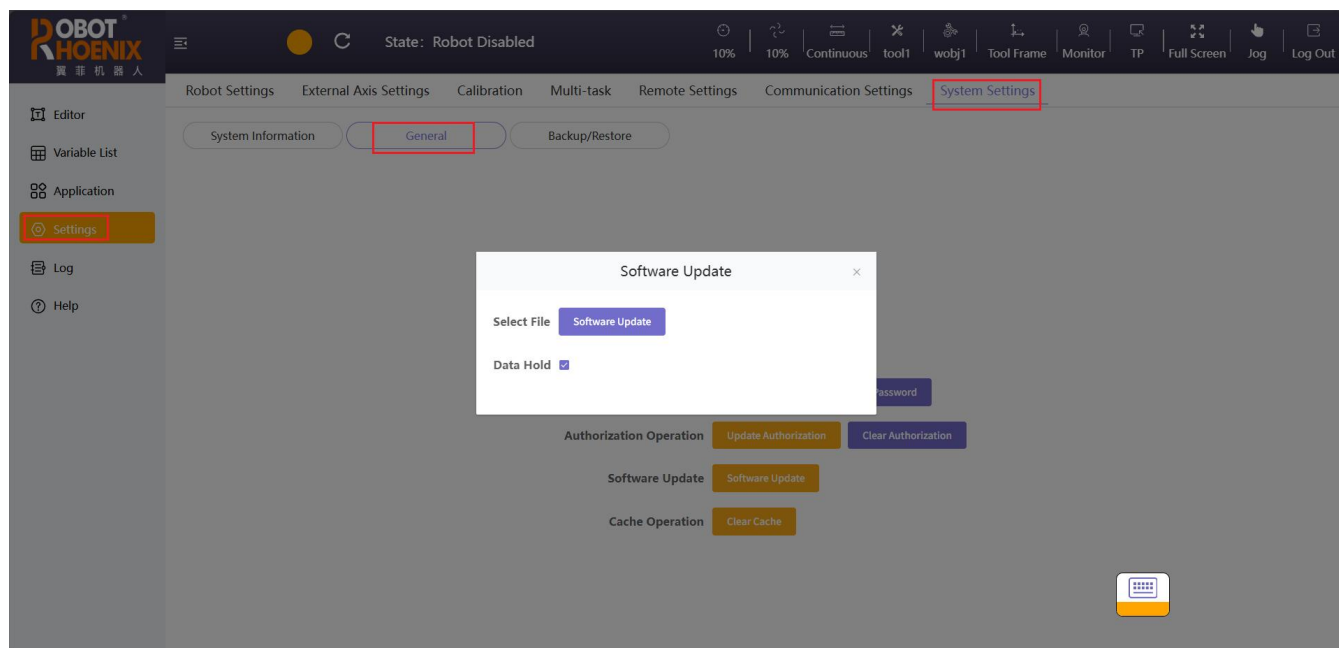


2.8.2.6. Software Update

You need to switch to “Administrator” or “Manufacturer” permission.

Operation Method

Click “Settings” → “System Settings” → “General” → “Software Update”. In the pop-up Software Update interface, choose whether to retain data, click the “Software Update” button, select the update file, and then click OK. As shown in the figure below:



Note:

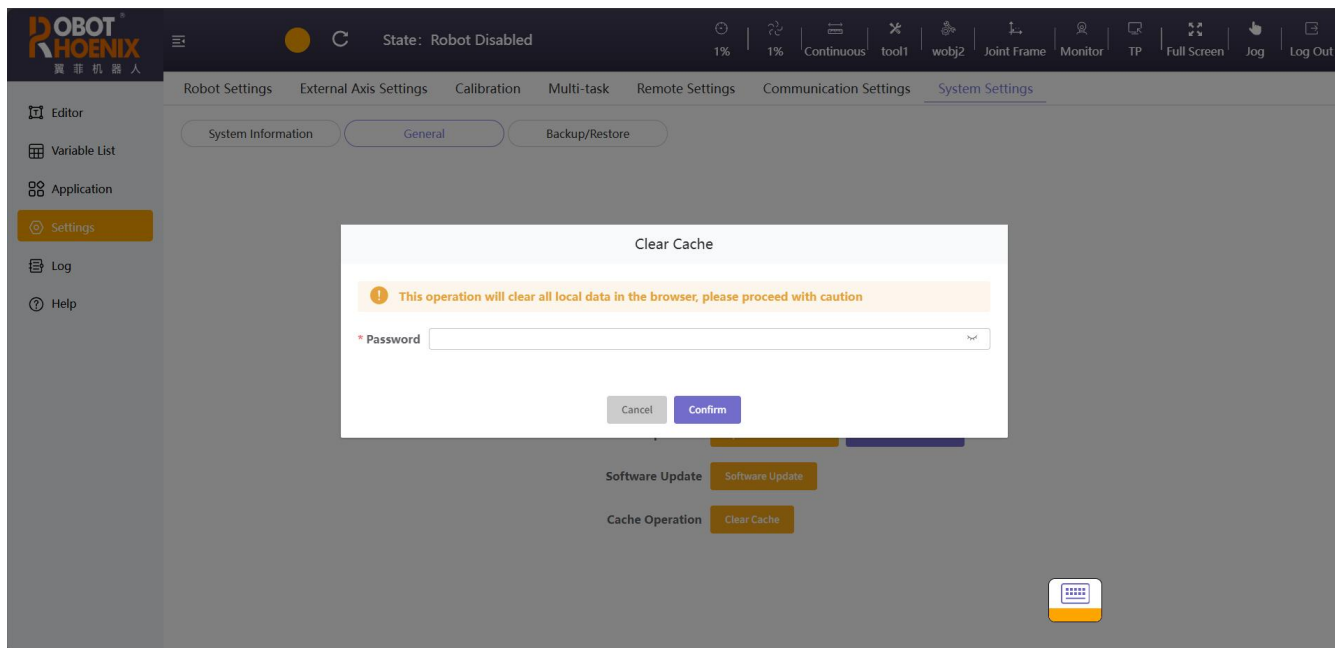
The Software Update function cannot retain existing data when updating across major versions (e.g., from 1.16.x to 1.17.x). If needed, you can download program files or variable tables from the original version, and re-upload them after the update is complete.

2.8.2.7. Clear Cache

You need to switch to “Manufacturer” permission.

Operation Method

To clear the browser cache, switch to “Manufacturer” permission, then click “Settings” → “System Settings” → “General” → “Clear Cache”. In the pop-up Clear Cache interface, enter the password and click OK. This is generally used after performing a software update. As shown in the figure below:



2.8.3. Backup and Restore

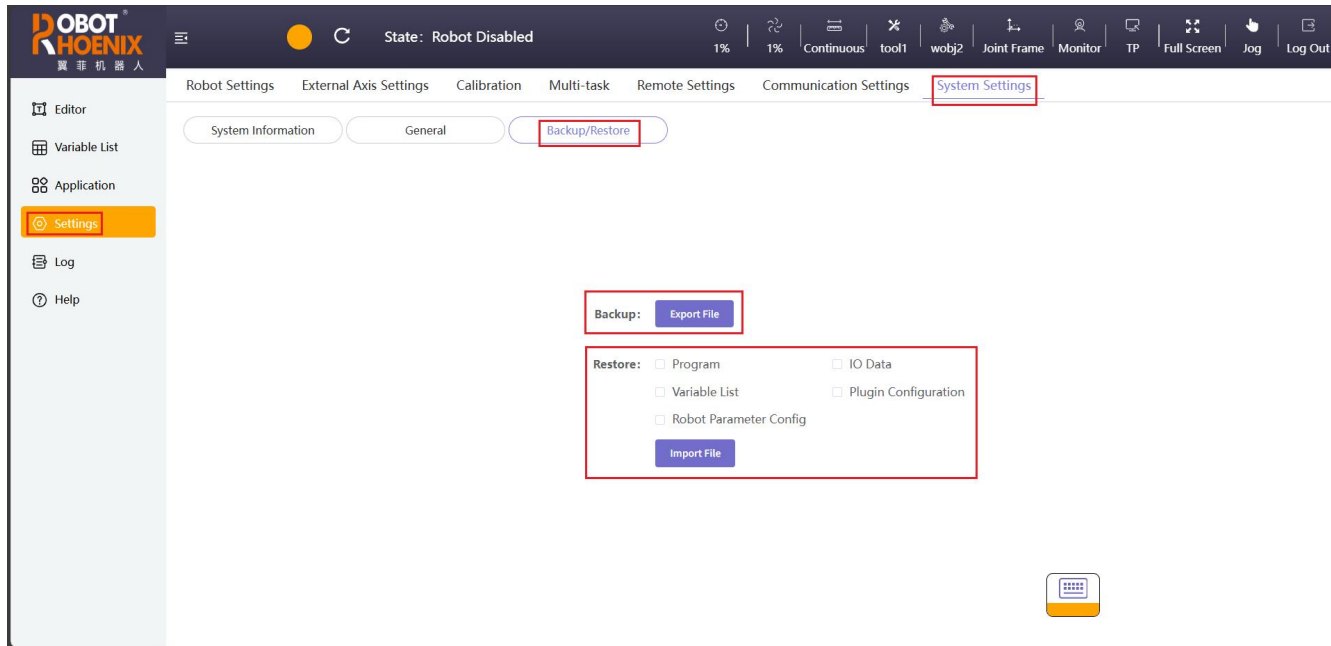
The Gorilla Studio robot control system supports backup and restore of program files, variable tables, robot parameter configurations, I/O data, and plugin configurations.

You need to switch to “Administrator” or “Manufacturer” permission.

Operation Method

To perform a backup, click the “Export File” button to download the file. Do not refresh the

page or perform other operations during the backup process. As shown in the figure below:



To perform a restore, select the desired options by checking “Program Files”, “Variable Tables”, “Robot Parameter Configurations”, “I/O Data”, or “Plugin Configurations”, then click the “Import File” button. Choose the backup file to open, and the restore process will begin.

Note:

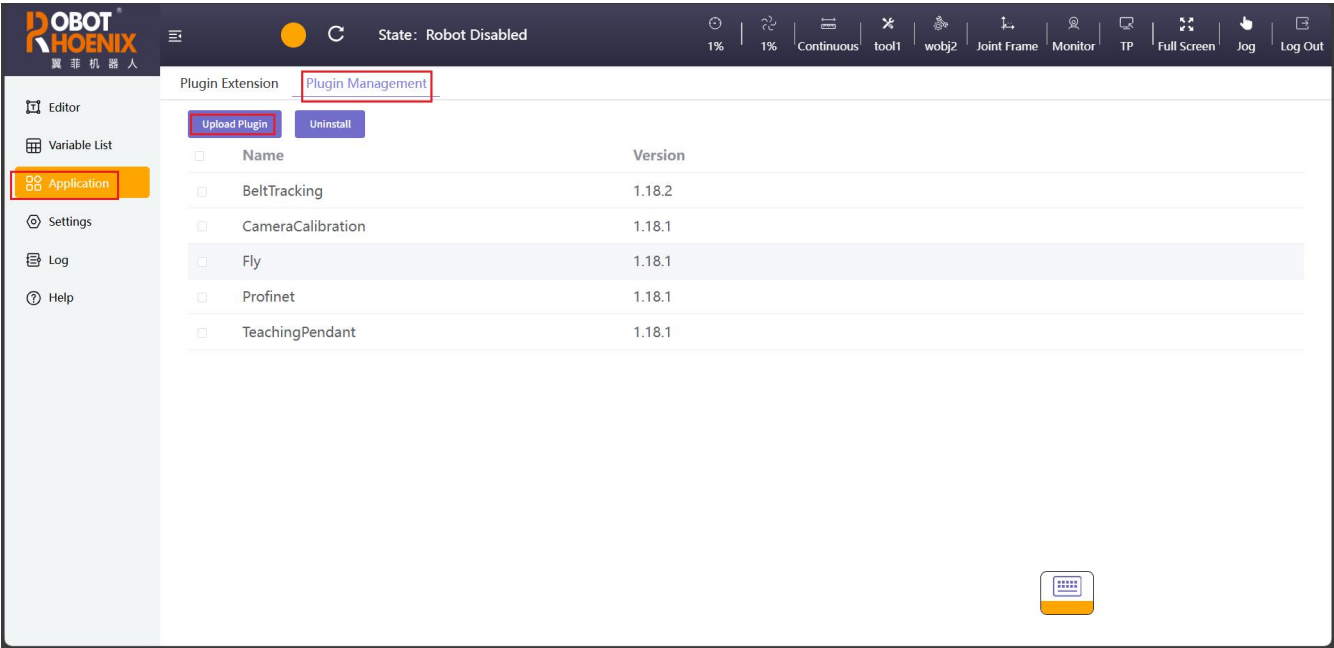
Restoration must be performed when the robot is in the “Robot Not Enabled” state. Backup files from versions with a large version gap (e.g., from 1.16.x to 1.17.x) are not supported for restoration. If needed, you can download program files or variable tables from the original version and re-upload them after completing the update.

2.8.4. Plugin Management

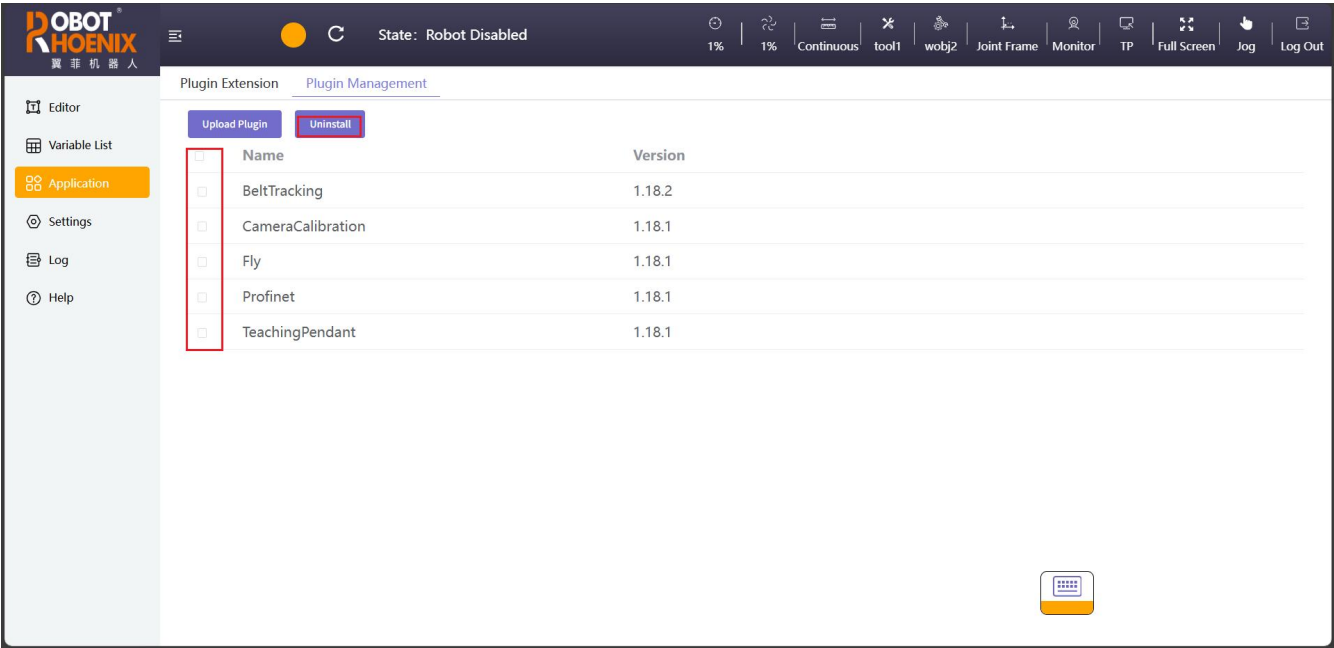
The Gorilla Studio robot control system supports uploading and uninstalling plugins.

Operation Method

To upload a plugin, switch to Manufacturer permission, then click “Application” → “Plugin Management” → “Upload Plugin”. This will allow you to upload the plugin. As shown in the figure below:



To uninstall a plugin, select the plugin you want to remove and click the “Uninstall” button.



3. Variable Table

3.1. Global Variables

3.1.1. Overview

Variables support an array format.

Global variables are valid in all program files and are highlighted in red.

Within a program, the attribute values of variables such as tool1–tool16, user1–user16, wobj1–wobj16, vel1–vel8, acc1–acc8, and zone1–zone8 can be assigned to variables of the same type, but their values cannot be directly modified.

The Global Variable Management interface allows you to add, view, search, upload, download, annotate, copy, modify, and delete global variables within the robot system. The currently supported variable types include:

Type	Variable	Description
Basic Type	Int	Integer: The allowed value range is -2147483647 to 2147483647.
	Double	Double-precision float: The allowed value range is -999999999999.999 to 999999999999.999.
	Float	Single-precision float type, with an allowable value range from -999999999999.999 to 999999999999.999.
	Bool	Boolean type, with values of either True or False.
	String	String type, allowing up to 1000 characters.
	Word	Unsigned short integer, with an allowable range from 0 to 65535.
Complex Type	Vel	Speed, with an internal format of (tcp, joint, rotation, jumpUp, jumpDown). tcp (%): Cartesian space speed percentage, with an allowable range from 1 to 100.

		joint (%): Joint space speed percentage, with an allowable range from 1 to 100. rotation (%): Cartesian rotation speed percentage, with an allowable range from 1 to 100. jumpUp (%): Ascending segment speed. The speed percentage for the ascending segment of the MoveLJump command, with an allowable range of 1% to 100%. jumpDown (%): Descending segment speed. The speed percentage for the descending segment of the MoveLJump command, with an allowable range of 1% to 100%.
	Acc	Acceleration, with an internal format of (tcp, joint, rotation, jumpUp, jumpDown).. tcp (%): Cartesian space acceleration percentage, with an allowable range from 1 to 100. joint (%): Joint space acceleration percentage, with an allowable range from 1 to 100. rotation (%): Cartesian rotation acceleration percentage, with an allowable range from 1 to 100. jumpUp (%): Ascending segment acceleration. The acceleration percentage for the ascending segment of the MoveLJump command, with an allowable range of 1% to 100%. jumpDown (%): Descending segment acceleration. The acceleration percentage for the descending segment of the MoveLJump command, with an allowable range of 1% to 100%.
	Zone	Zone, with an internal format of (distance, percent). distance (mm): Cartesian space zone distance, with an allowable range from 0 to 1999. percent (%): Joint space zone percentage, with an allowable

		range from 0 to 100.
	Pose	Pose, with an internal format of (X, Y, Z, Rx, Ry, Rz). X (mm) : The allowable value range is -9999 to 9999. Y (mm) : The allowable value range is -9999 to 9999. Z (mm) : The allowable value range is -9999 to 9999. Rx (°) : The allowable value range is -9999 to 9999. Ry (°) : The allowable value range is -9999 to 9999. Rz (°) : The allowable value range is -9999 to 9999.
	AxisPos	Axis Position, internal format: (J1, J2, J3, J4, J5, J6). J1 (°) : 1 Axis Position, The allowable value range is -9999 to 9999. J2 (°) : 2 Axis Position, The allowable value range is -9999 to 9999. J3 (°) : 3 Axis Position, The allowable value range is -9999 to 9999. J4 (°) : 4 Axis Position, The allowable value range is -9999 to 9999. J5 (°) : 5 Axis Position, The allowable value range is -9999 to 9999. J6 (°) : 6 Axis Position, The allowable value range is -9999 to 9999.
	RobotTarget	Internal format: (pose, confData, eAxisPos, tool, wobj) ● pose (Target Pose) : X (mm) : The allowable value range is -9999 to 9999. Y (mm) : The allowable value range is -9999 to 9999. Z (mm) : The allowable value range is -9999 to 9999. Rx (°) : The allowable value range is -9999 to 9999.

		Ry (°) : The allowable value range is -9999 to 9999. Rz (°) : The allowable value range is -9999 to 9999. ● confData (Configuration) : 1 Axis Rotation Quadrant: The allowable value range is -4 to 3. 2 Axis Rotation Quadrant: The allowable value range is -4 to 3. 3 Axis Rotation Quadrant: The allowable value range is -300 to 300. 4 Axis Rotation Quadrant: The allowable value range is -300 to 300. 5 Axis Rotation Quadrant: The allowable value range is -4 to 3. 6 Axis Rotation Quadrant: The allowable value range is -4 to 3. 7 Axis Rotation Quadrant: The allowable value range is -4 to 3. Pose Type: The allowable value range is 0 to 8. ● eAxisPos (External Axis Position) : E1(mm): External Axis 1 Axis Position, The allowable value range is -9999 to 9999. E2(mm): External Axis 2 Axis Position, The allowable value range is -9999 to 9999. E3(mm): External Axis 3 Axis Position, The allowable value range is -9999 to 9999. E4(mm): External Axis 4 Axis Position, The allowable value range is -9999 to 9999. E5(mm): External Axis 5 Axis Position, The allowable value
--	--	--

		range is -9999 to 9999. E6(mm): External Axis 6 Axis Position, The allowable value range is -9999 to 9999. ● tool(Tool Coordinate System): The allowable value range is tool1 to tool16. ● wobj (Work object coordinate system) : The allowable value range is wobj1 to wobj16
	JointTarget	Internal format: (jointPos, eAxisPos) ● jointPos (Axis Position) : J1 (°) : 1 Axis Position, The allowable value range is -180 to 180. J2 (°) : 2 Axis Position, The allowable value range is -180 to 180. J3 (°) : 3 Axis Position, The allowable value range is -9999 to 9999. J4 (°) : 4 Axis Position , The allowable value range is -999999999 to 999999999 J5 (°) : 5 Axis Position, The allowable value range is -360 to 360 J6 (°) : 6 Axis Position, The allowable value range is -360 to 360 ● eAxisPos (External Axis Position) : E1(mm): External Axis 1 Axis Position, The allowable value range is -9999 to 9999. E2(mm): External Axis 2 Axis Position, The allowable value range is -9999 to 9999. E3(mm): External Axis 3 Axis Position, The allowable value range is -9999 to 9999. E4(mm): External Axis 4 Axis Position, The allowable value

		range is -9999 to 9999. E5(mm): External Axis 5 Axis Position, The allowable value range is -9999 to 9999. E6(mm): External Axis 6 Axis Position, The allowable value range is -9999 to 9999.
	JumpData	Internal format: (upDis, upPer, downDis, downPer, rotUpDis, rotDownDis) upDis (Ascending Distance) (mm): The ascending distance for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999. upPer (Ascending Zone Percentage) (%): The percentage of the ascending zone in a door-shaped motion. The allowable range is 0 to 100. downDis (Descending Distance) (mm): The descending distance for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999. downPer (Descending Zone Percentage) (%): The percentage of the descending zone in a door-shaped motion. The allowable range is 0 to 100. rotUpDis (Pose Start Height) (mm): The pose start height for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999. rotDownDis (Pose Stop Height) (mm): The pose stop height for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The

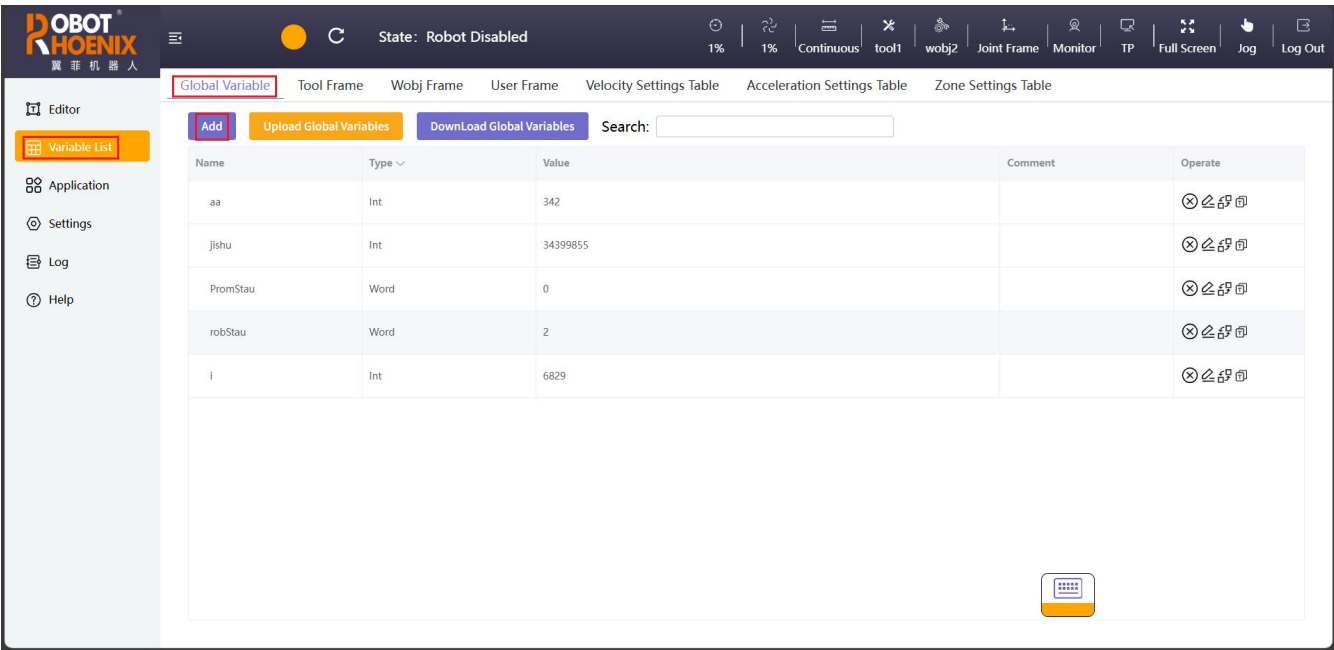
		allowable range is -999 to 999.
--	--	---------------------------------

3.1.2. Edit Variable

3.1.2.1. Add

Operation Method

To add a global variable, click the “Add” button. Expand the “Type” dropdown and select the variable type you want to add. Enter the variable name and select the dimension: “None” represents a single variable, and “1” represents a one-dimensional array variable (array length must be specified, valid range 1–1024). Enter the initial value of the variable (for RobotTarget and JointTarget types, you can obtain the current value by clicking the teach button), add any comments, and click OK. As shown in the figure below:



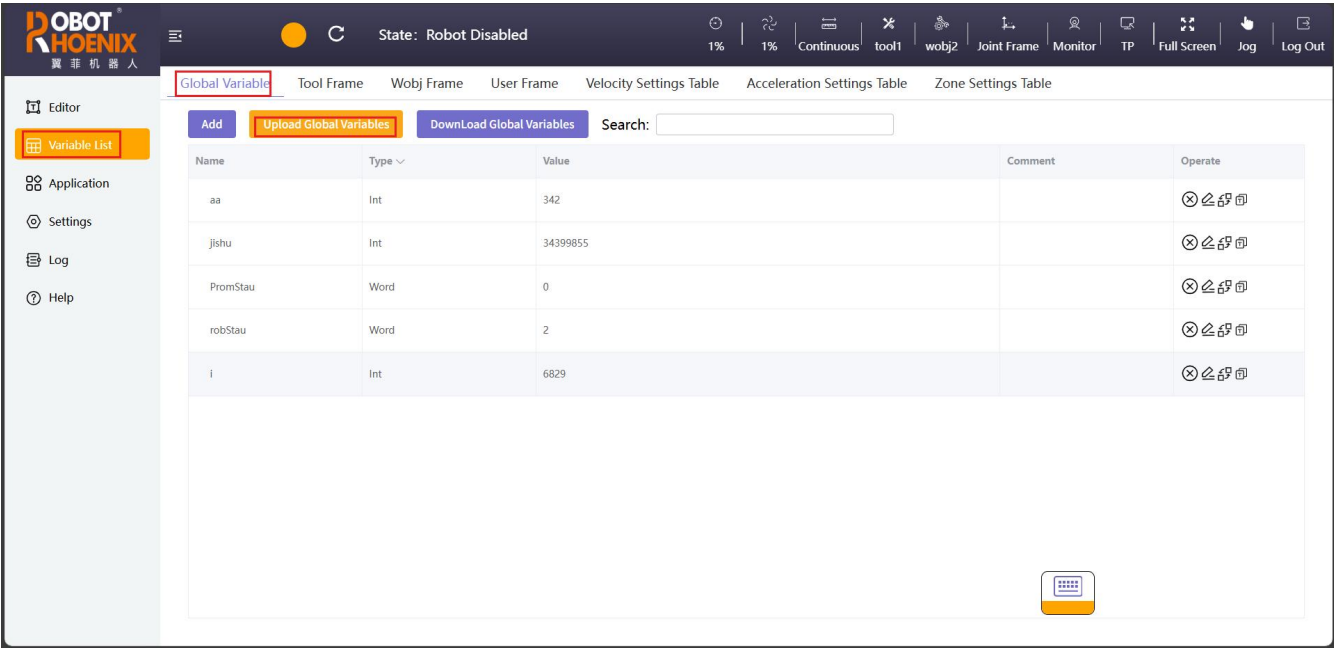
Note:

- Variable names can consist of letters, underscores, and numbers, and must start with a letter or underscore. However, variable names cannot be the same as system keywords.
- Local variable names cannot be the same as global variable names.
- Within the same program, duplicate local variable names are not allowed.
- Across different programs, duplicate local variable names are allowed.

3.1.2.2. Upload Global Variables

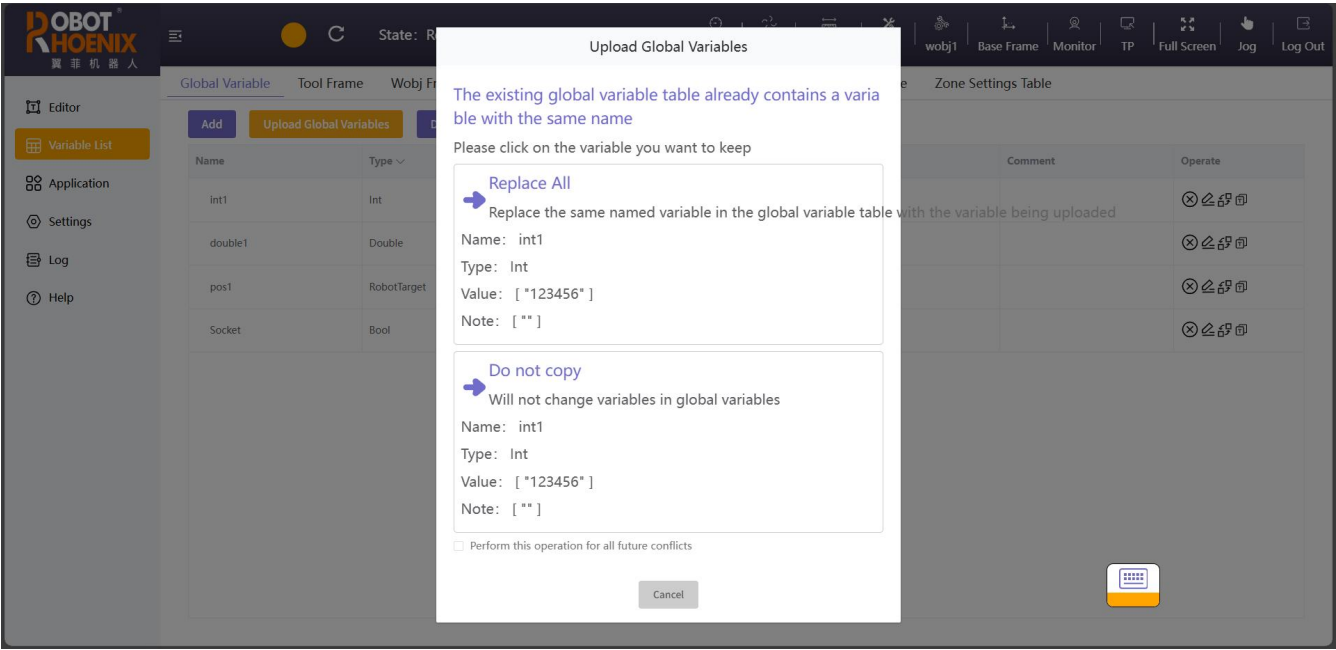
Operation Method

To upload global variables, click the “Upload Global Variables” button and select the file you want to upload. As shown in the figure below:



Note:

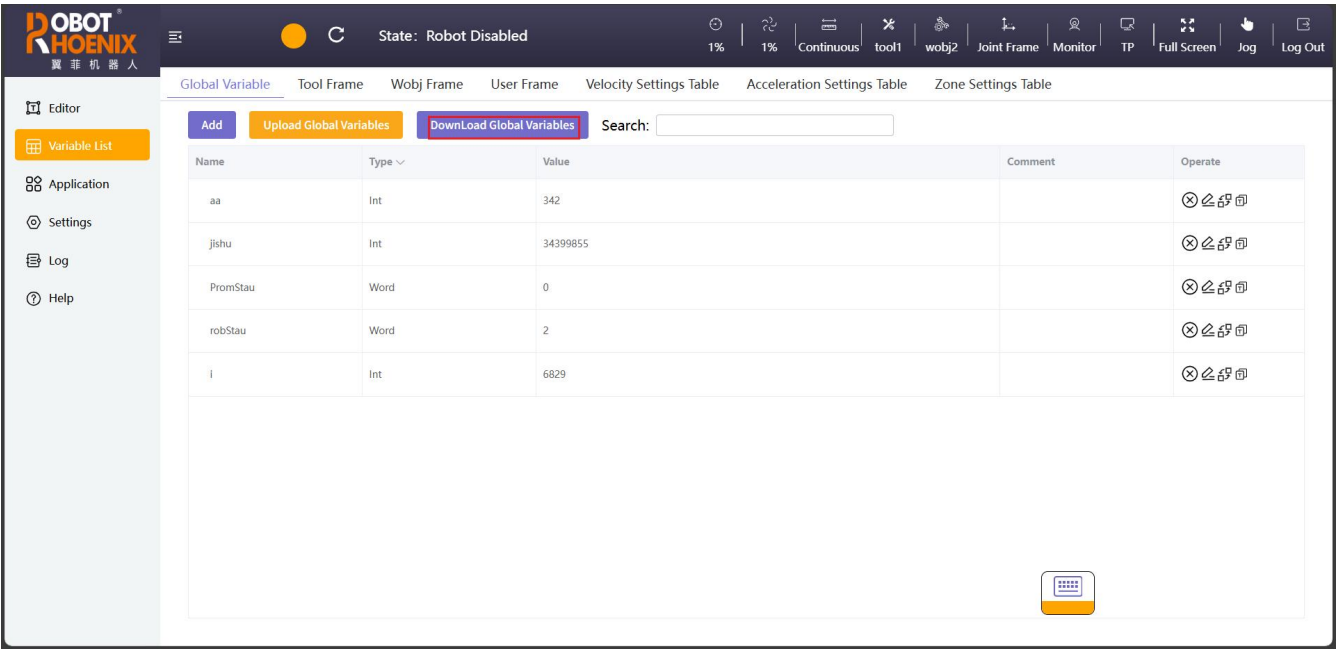
If the existing global variable table already contains variables with the same name, choose “Copy and Replace” (replace the variables in the global variable table with the ones being uploaded) or “Do Not Copy” (keep the existing variables in the global variable table). As shown in the figure below:



3.1.2.3. Download Global Variables

Operation Method

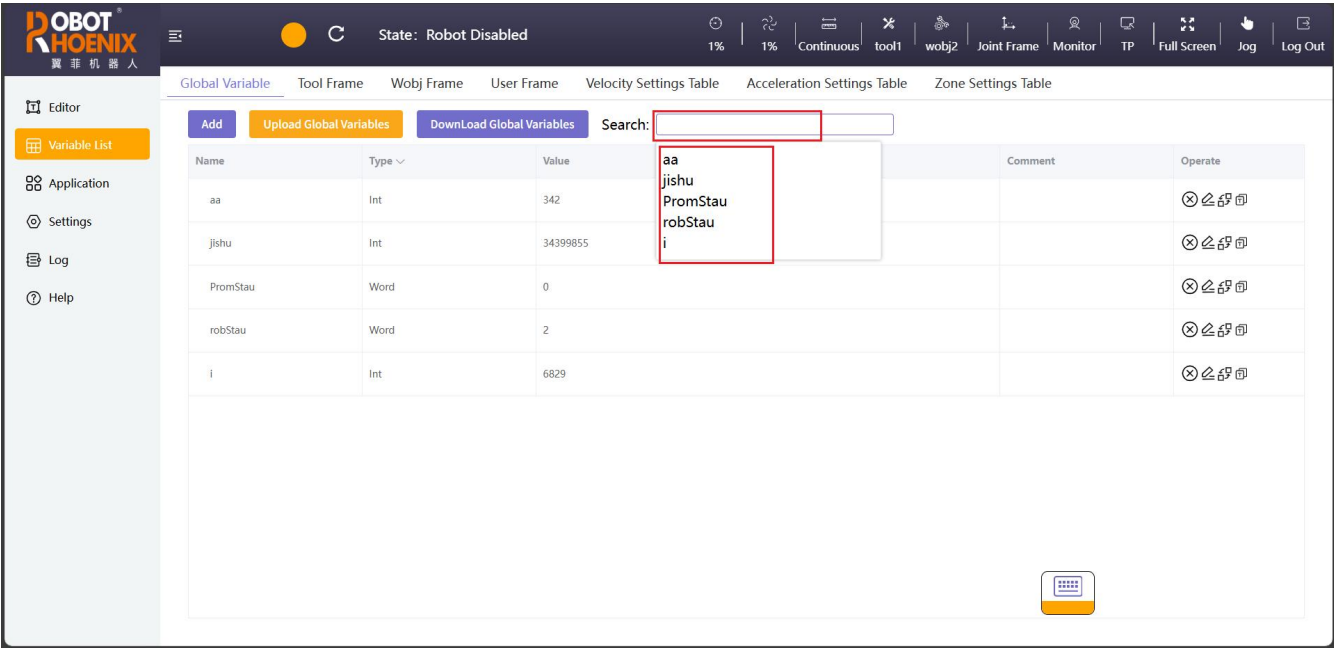
To download global variables, click the “Download Global Variables” button to download the global variable table file. As shown in the figure below :



3.1.2.4. Search

Operation Method

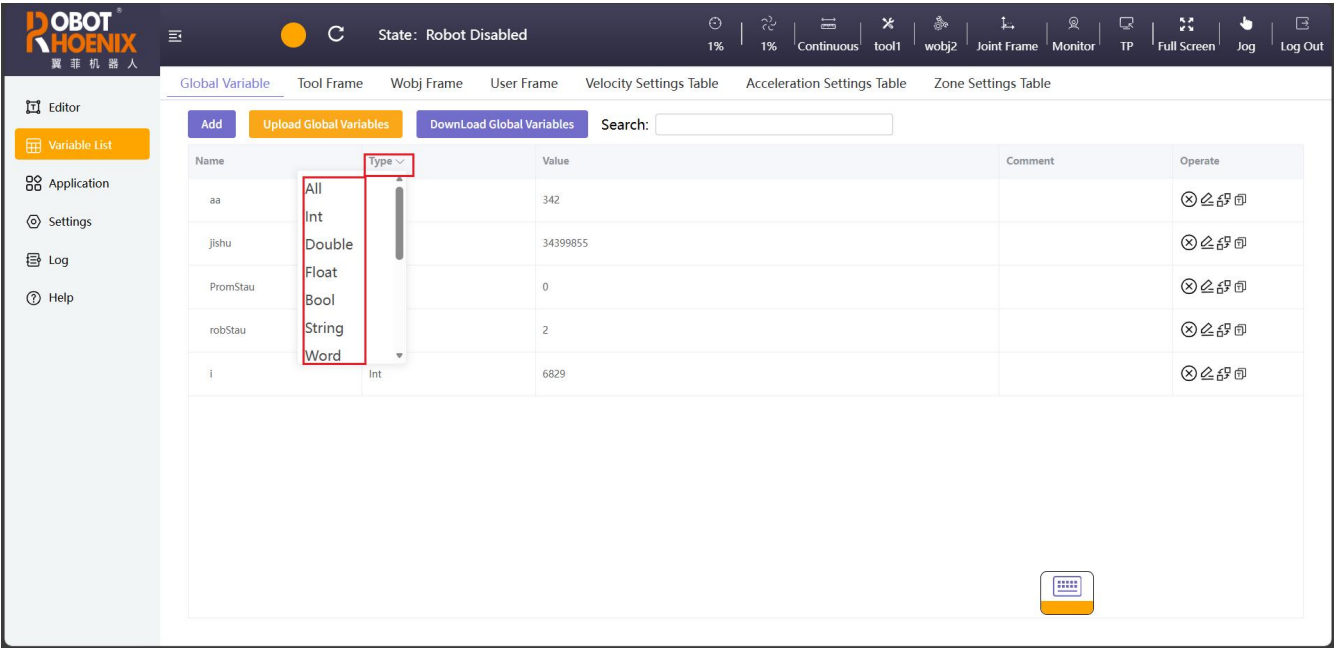
To search for a global variable, click the “Search” input box and either type in the variable name manually or select it from the dropdown options. As shown in the figure below:



3.1.2.5. Filter

Operation Method

To filter global variables by type, click the “Type” dropdown arrow and select the variable type from the dropdown list. As shown in the figure below:



3.1.2.6. Delete

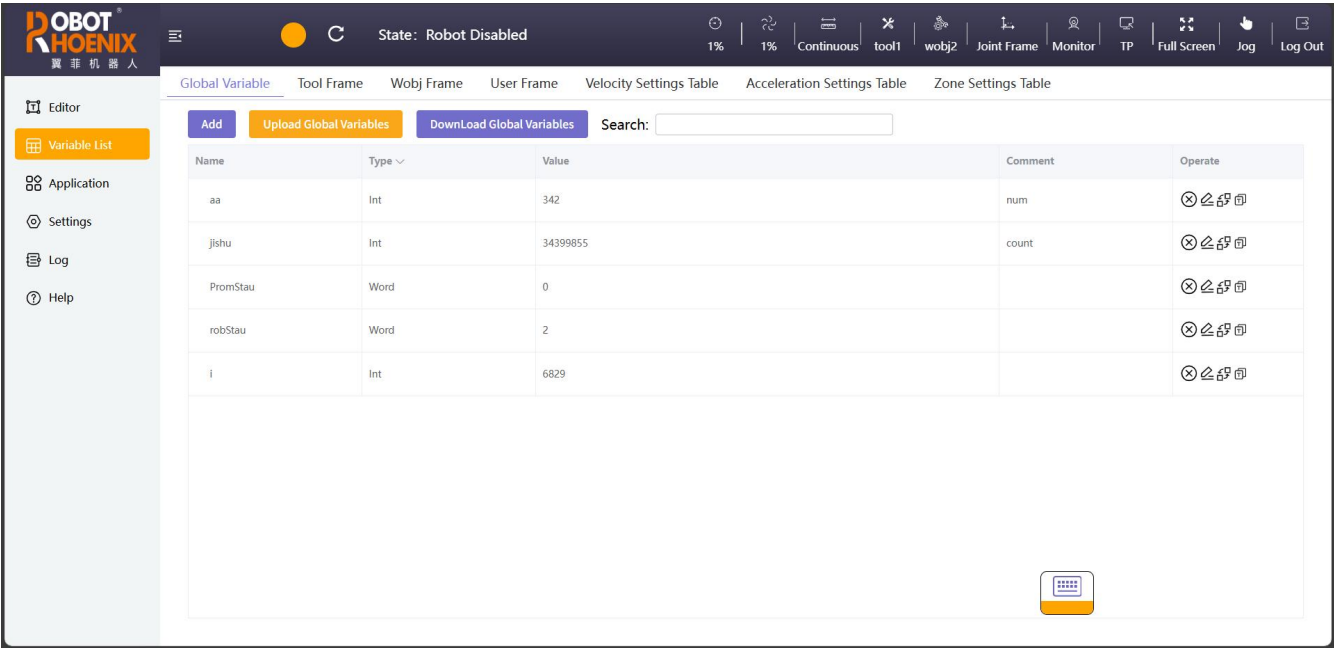
Operation Method

To delete a global variable, click the variable and then click the Delete button (⊗) in the action bar. In the popup confirmation window, click OK.

3.1.2.7. Modify Value and Comment

Operation Method

To modify a global variable, click the variable and then click the Edit button (✎) in the action bar. In the popup window, you can modify the variable name, variable value, or add comments. Click OK to confirm.



3.1.2.8. Modify Variable Definition

Operation Method

To modify a global variable's type, name, dimension, or array length, click the variable and then click the Modify button () in the action bar. In the popup window, make the desired changes and click OK to confirm.

Note:

Performing this operation will affect the original data, so please proceed with caution.

3.1.2.9. Copy

Operation Method

To copy the name of a global variable, click the variable and then click the Copy button () in the action bar. You can then use the Paste button in the program editing interface to paste it into the program editing area.

3.2. Tool Coordinate System

3.2.1. Tool

Definition

A **tool** refers to the device installed on the robot's end flange to perform specific processing tasks. Common tools include pneumatic/electric grippers, suction cups, welding guns, spray nozzles, and others. Robots are not shipped with any tools; you need to select a suitable tool (either purchased externally or self-designed) and complete its installation and setup before using the robot. Before using any tool, it must be **calibrated** to obtain the TCP (Tool Center Point) data.

Description

Tool-type variables are used to record tool parameters, including the TCP, posture, and kinematic parameters of the tool used by the robot. When the robot interacts with the external environment using a tool, the Tool variable affects the robot's motion in the following ways:

- Only the Tool Center Point (TCP) moves according to the programmed path and speed. When the robot performs a pure spatial rotation, only the TCP remains fixed.
- The programmed motion path and speed are defined as the movement of the Tool Coordinate System relative to the Work Object Coordinate System. Therefore, replacing the tool or workpiece with a well-calibrated one does not affect the path shape or speed magnitude.
- When using an external tool, the programmed speed refers to the speed of the workpiece.

Internal Format

The internal format of a Tool-type variable is: (Frame, Hold, Load)

- Frame

Records the Tool Coordinate System (Tool CS) in use, including:

- (1) TCP represents the offset in the X, Y, and Z directions relative to the robot's end flange coordinate system, unit: mm.
- (2) Tool Coordinate System represents the orientation offset relative to the flange coordinate system.

- **Hold**

Defines whether the tool is installed on the robot. True indicates the tool is installed on the robot, while False indicates the tool is not installed, and an External tool is currently in use.

When performing jog operations or executing programs, within the combination of tools and workpieces in use, only one Hold parameter can be True at a time. That is, if the tool's Hold is True, the corresponding workpiece Hold must be False, and vice versa.

- **Load**

Records the tool's dynamic parameters. For a standard tool, Load describes the dynamics of the entire tool. For an External tool, Load describes the dynamics of the gripper that the robot uses to hold the workpiece.

For a standard tool installed on the robot, its Load parameters include:

- (1) Tool mass, unit: kg.
- (2) Tool center of mass, described in the flange coordinate system, unit: mm.
- (3) Orientation of the principal axes of inertia, described in the flange coordinate system.
- (4) Moment of inertia along the principal axes, unit: $\text{kg}\cdot\text{m}^2$. If all inertia components are defined as 0 $\text{kg}\cdot\text{m}^2$, the tool will be treated as a point mass.

3.2.2. Tool Center Point

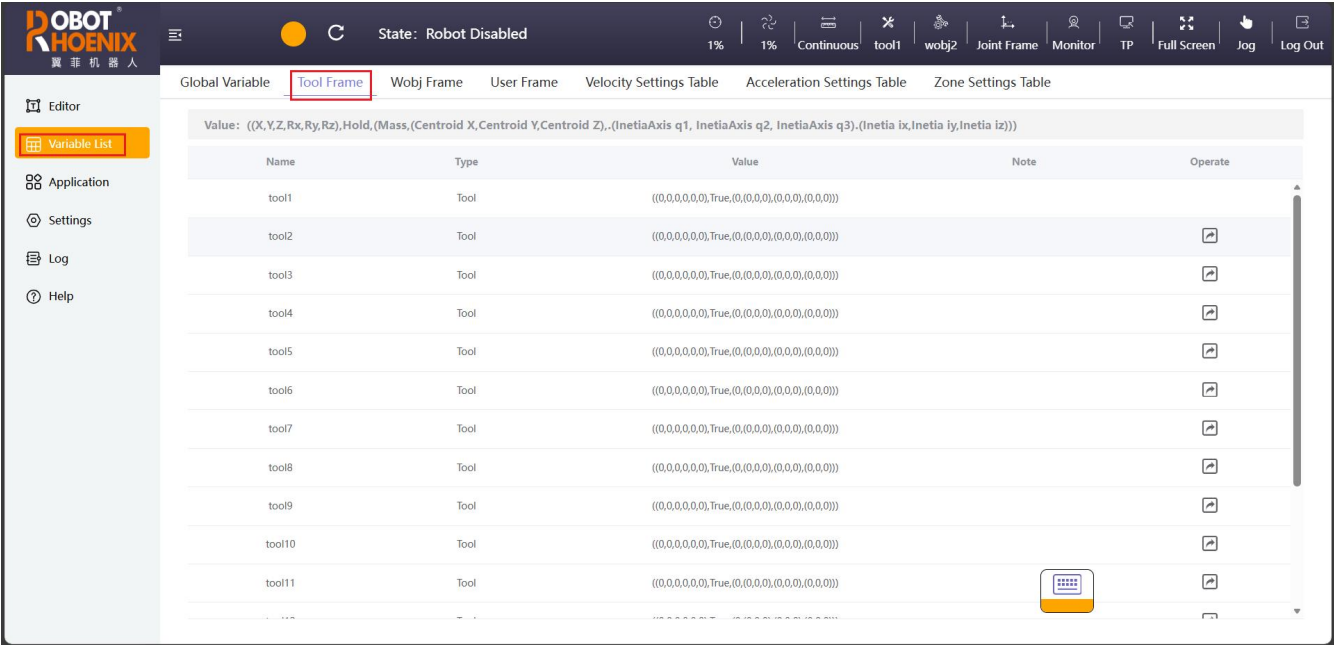
Definition

The Tool Center Point (TCP) is a specific point located on the tool, which the robot typically uses as the reference for performing operations—for example, the tip of a welding wire on a welding torch or the fingertip of a pneumatic gripper. The robot can rotate and change its orientation around the TCP while keeping the TCP position fixed. Different tools have different TCPs, and defining an appropriate TCP according to the actual situation can greatly improve programming efficiency. The TCP also serves as the origin of the Tool Coordinate System. For more details, refer to the Tool variable description.

Explanation

Unless otherwise specified, wherever this manual mentions “robot position, speed, or acceleration,” it refers to the position, speed, and acceleration of the TCP relative to the Work Object Coordinate System.

3.2.3. Tool Coordinate System



Explanation

Tool Coordinate System (TCS) calibration refers to the process of measuring the position and orientation offsets of the Tool Coordinate System relative to the robot flange coordinate system. If the tool manufacturer provides these offset data, you can select the “Manual Input” option in the TCS calibration interface to directly enter the values without performing the calibration process. For tools without dimensional data, the calibration method provided by the Gorilla Studio robot control system must be used. For coordinate systems other than tool1, you can click the corresponding jump button  to directly navigate to the calibration interface.

Calibrating the Tool Coordinate System

Before calibrating the Tool Coordinate System, a fixed External tip point needs to be prepared. This point should be within the robot’s working range, and the tool to be calibrated must be able to reach it with a reasonably flexible posture. For detailed operation methods, please refer to “Calibration” → “Tool Coordinate System Calibration.”

Note:

The Gorilla Studio robot control system predefines 16 tool variables. Among them, Tool1 cannot be modified; its Tool Coordinate System coincides with the flange coordinate system, and all dynamic parameters are set to 0.

Modifying a tool definition can only be done through the coordinate system calibration interface. For detailed steps, please refer to [Calibrating Tool Coordinate System](#). In the variable table interface, Tool-type variables can only be viewed—they cannot be created or modified.

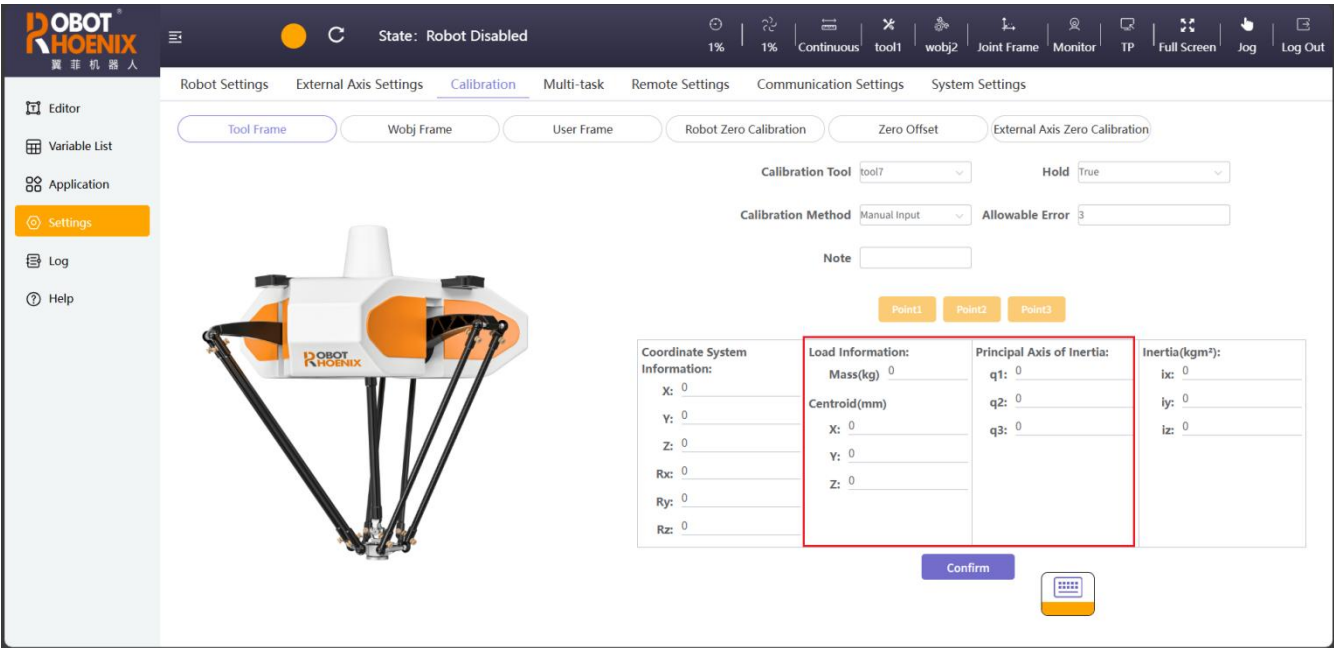
Warning

Please do not attempt to directly modify Tool-type variables in the program editing area, as this may cause unpredictable errors. If modification is necessary, please use the calibration interface. For detailed instructions, refer to **Tool Coordinate System Calibration**.

3.2.4. Tool Load Parameters

Explanation

As mentioned earlier, fully defining a tool requires specifying both its kinematic and dynamic parameters. Using the Tool Coordinate System calibration only determines the tool's kinematic parameters. The tool's dynamic parameters need to be specified separately and can be manually entered in the Tool Coordinate System calibration interface by selecting the manual input method, as shown below:



3.2.5. External Tool

Definition

Under normal circumstances, the tool is installed on the robot and moves with the robot to

perform the designated task. Such tools are referred to as standard tools, with common examples including grippers, suction cups, and welding guns. However, in certain specific situations, installing the tool on the robot may affect normal operation, for example:

(1) The tool to be used is relatively large or heavy, making it inconvenient to install on the robot or potentially affecting the robot's motion performance.

(2) The workpiece to be processed is relatively large, and the robot's normal working range cannot adequately cover it.

(3) Certain specific process actions need to be performed; for example, when polishing a square object, the robot needs to perform self-rotation around each of the four corners.

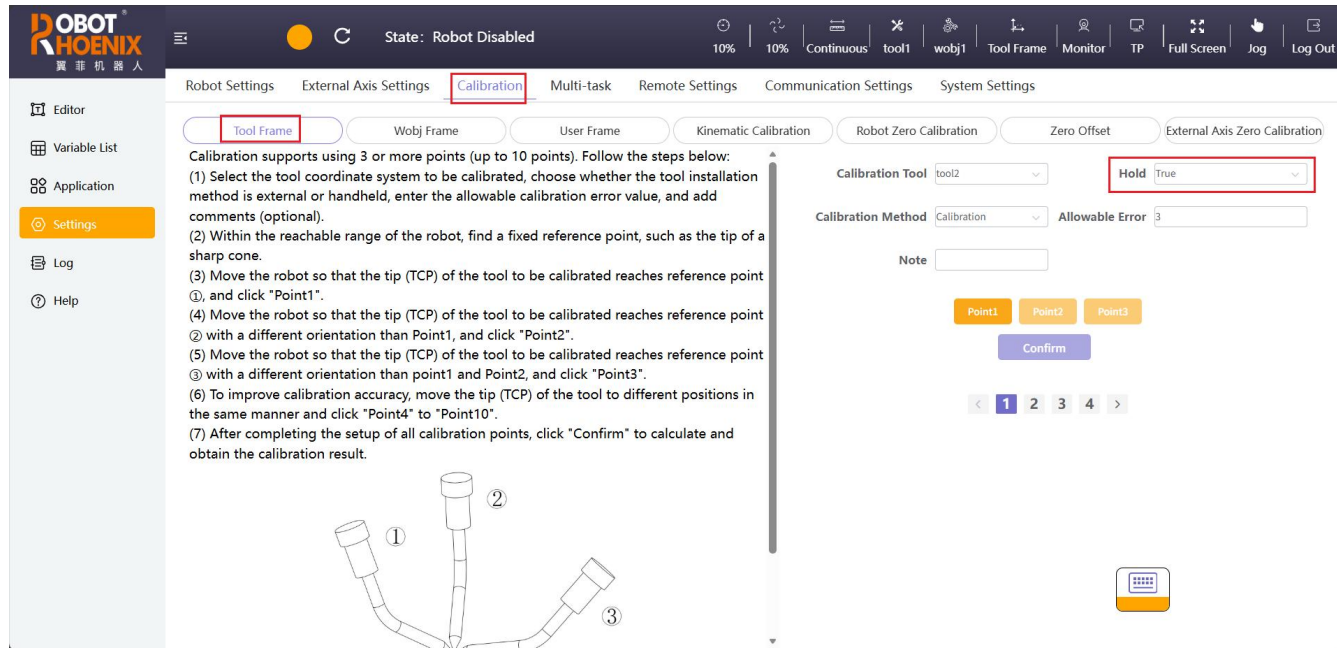
In these situations, it is more appropriate and convenient to mount the workpiece on the robot while keeping the tool fixed at a certain position externally. We refer to such tools, which are installed outside the robot and remain stationary at a fixed location, as External tools.

Create External Tool

External tools are also described using Tool-type variables. Within a Tool-type variable, a dedicated flag called Hold is used to define whether a tool is a standard tool or an External tool. Creating an External tool is very simple: in the Tool calibration interface, select the desired tool and set Hold to False.

Calibrating the External Tool Coordinate System

As shown in the figure below, click the left-side navigation bar: Settings → Calibration → Tool Coordinate System to enter the Tool Coordinate System calibration interface for calibration.



The calibration method for an External tool is the same as for a standard tool. However, calibrating the External Tool Coordinate System requires using a pre-calibrated standard tool. Here, the four-point method is used as an example for illustration:

Step 1: Calibrate a standard tool with a tip, or select an already calibrated standard tool. The calibration accuracy should be as high as possible.

Step 2: Select the Tool Coordinate System that needs calibration, choose the “Four-Point Method” for calibration, set “Handheld” to False, fill in the allowed calibration error, and optionally add a comment.

Step 3: Move the robot so that the TCP of the calibrated standard tool points to the desired origin (O) of the External Tool Coordinate System, then click “Point 1”.

Step 4: Move the robot so that the TCP of the calibrated tool points to the origin of the desired Tool Coordinate System in different orientations, then confirm Point 2, Point 3, and Point 4 sequentially.

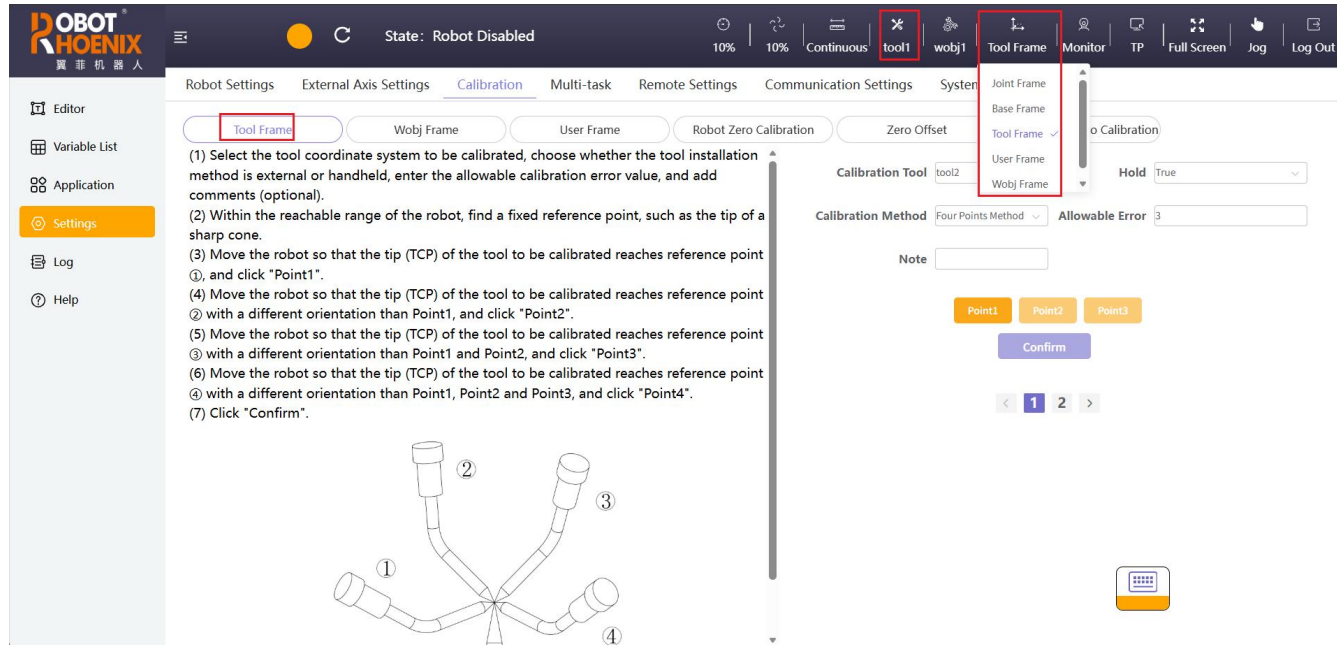
Step 5: Click “OK”.

3.2.6. Using Tools

Using Tools in Jog Mode

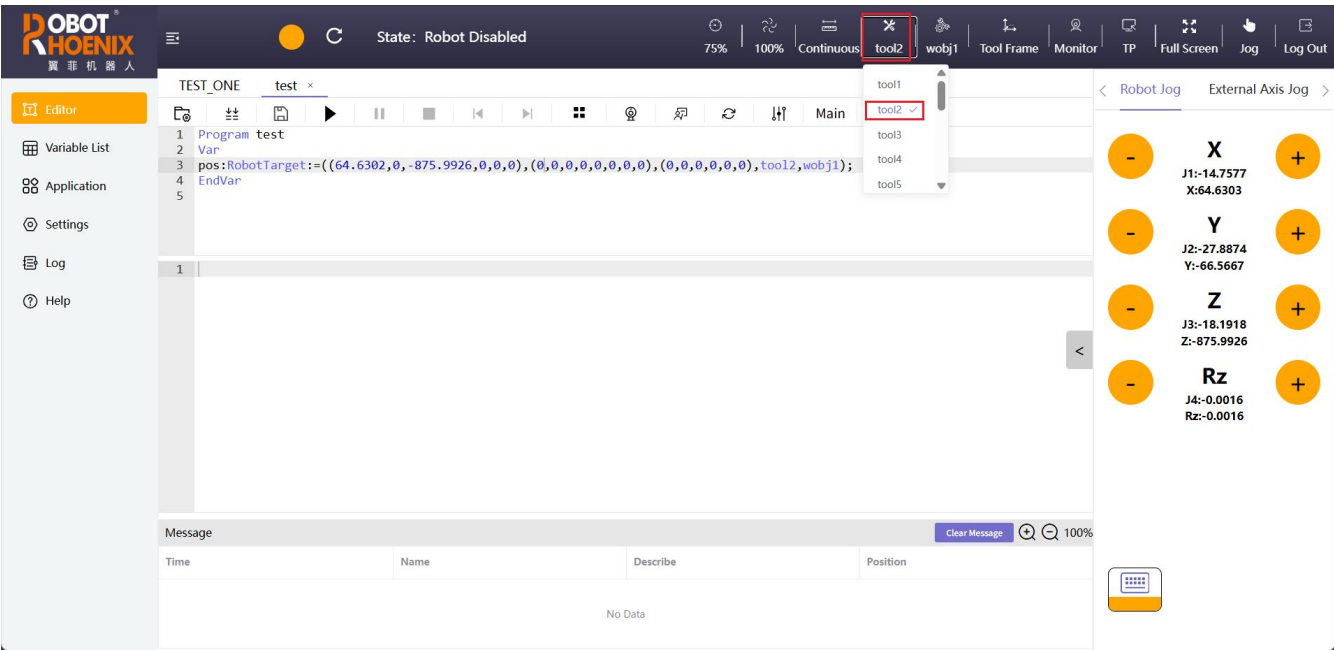
To use a specified tool for jog operations, click the Jog Coordinate System button () in the

toolbar at the top of the interface, select "Tool Coordinate System" from the dropdown menu; then click the Tool Coordinate System button (✂) to expand the dropdown list and choose the desired tool. As shown below:



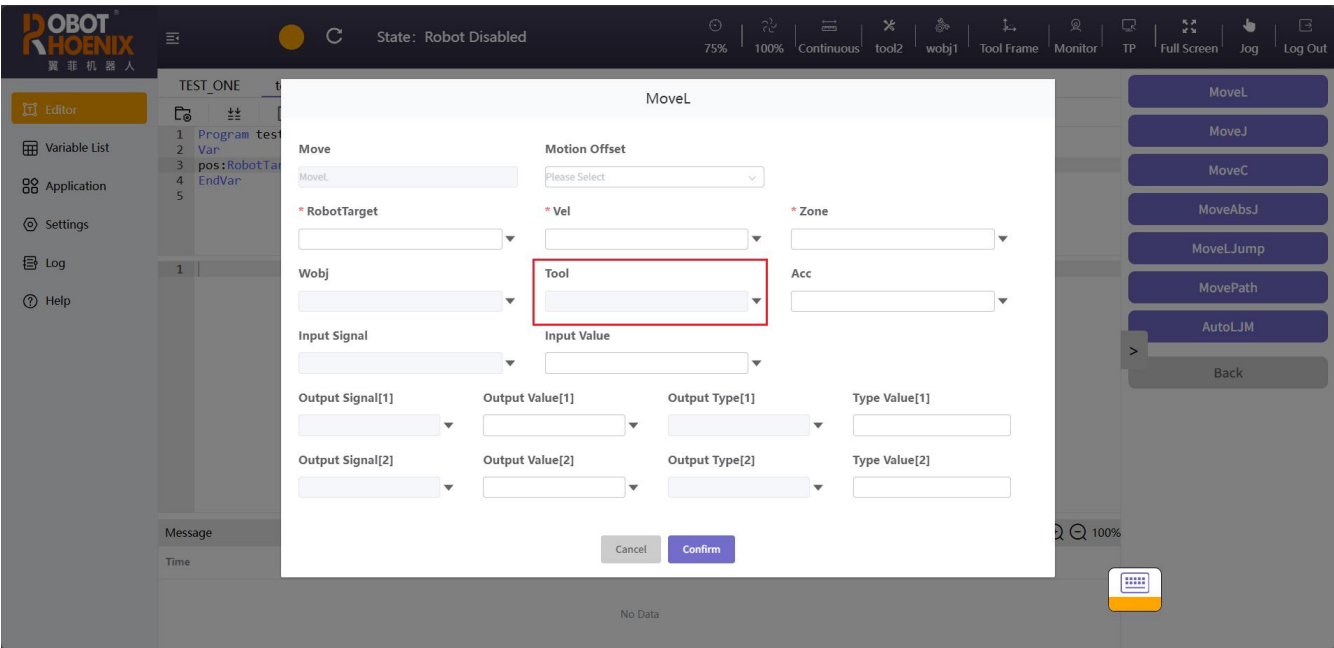
To use during teaching points

To use a specified tool when teaching local or global position variables, click the Tool Coordinate System button (✂) in the toolbar at the top of the interface, expand the dropdown menu, select the desired tool, and then proceed with the teaching. As shown in the figure below:



Use in the program

Using a specific tool in a program is very simple — just select the desired tool in the “Tool” parameter of the motion instruction. As shown below:



Note:

When inserting a motion instruction, if the tool parameter in the RobotTarget variable is not consistent with the currently selected tool, the motion will be executed according to the currently selected tool.

3.3. Work Object Coordinate System

3.3.1. Work Object

Definition

A workpiece refers to the item that the robot processes or handles using a tool. In the Gorilla Studio robot control system, a workpiece is described using a variable of the type Wobj (Work Object).

Explanation

Introducing the concept of a work object helps simplify programming steps and improve efficiency. All robot motion paths are defined relative to the Work Object Coordinate System, which offers two major advantages:

(1) When a workpiece is moved or when multiple identical workpieces are processed, you only need to recalibrate the Work Object Coordinate System. All programmed paths will update automatically without the need to rewrite the program.

(2) It allows processing of workpieces moved by an External Axis (such as a linear rail).

Each work object actually contains two coordinate systems:

A user coordinate system relative to the workpiece, which can be understood as the table or platform where the workpiece is placed. This is very useful when handling multiple identical workpieces.

A Work Object Coordinate System, which is fixed to the workpiece itself, and all program paths are defined relative to this coordinate system.

Internal Format

The internal format of a Wobj variable is: (Frame, Hold, User).

- Frame

Records the origin and orientation of the Work Object Coordinate System.

- Hold

Defines whether the workpiece is mounted on the robot. True indicates that the workpiece is installed on the robot and an External tool is currently in use; False indicates that the

workpiece is not installed on the robot and a standard tool is currently in use.

During jog or program execution, only one Hold parameter in the tool-workpiece combination can be True at a time. That is, if the tool's Hold is True, the corresponding workpiece's Hold must be False, and vice versa.

- User

User Coordinate System.

Note:

- When using a standard tool (non-External tool), the coordinate system definition chain is as follows:

Work object coordinate system is defined relative to the User Coordinate System;

User Coordinate System is defined relative to the Base Coordinate System.

As shown in the figure:

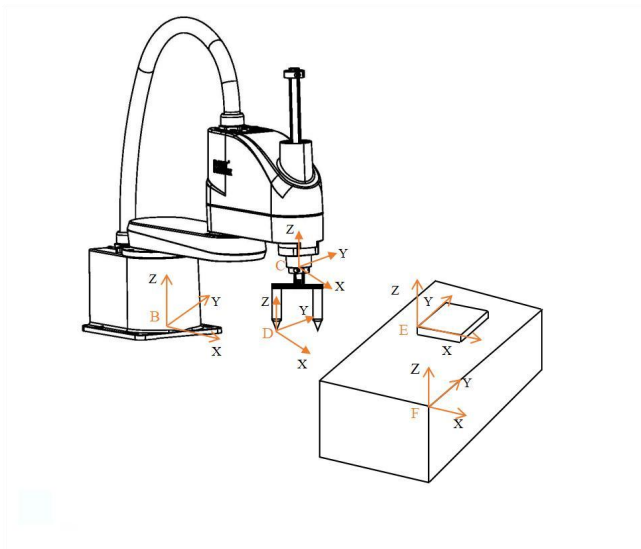
B = Base Coordinate System

C = Flange Coordinate System

D = Tool Coordinate System

E = Work Object Coordinate System

F = User Coordinate System



- When using an External tool, the coordinate system definition chain is as follows:

The Work Object Coordinate System is defined relative to the User Coordinate System.

The User Coordinate System is defined relative to the Flange Coordinate System, and the two coincide.

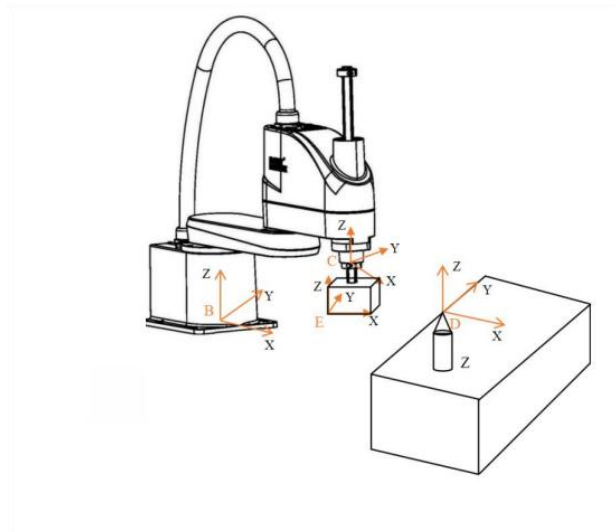
As shown in the figure:

B = Base Coordinate System

C = Flange Coordinate System

D = Tool Coordinate System

E = Work Object Coordinate System



3.3.2. Work Object Coordinate System

Name	Type	Value	Note	Operate
wobj1	Wobj	((0,0,0,0,0,0),False,user1)		
wobj2	Wobj	((0,0,0,0,0,0),False,user2)		
wobj3	Wobj	((0,0,0,0,0,0),False,user1)		
wobj4	Wobj	((0,0,0,0,0,0),False,user1)		
wobj5	Wobj	((0,0,0,0,0,0),False,user1)		
wobj6	Wobj	((0,0,0,0,0,0),False,user1)		
wobj7	Wobj	((0,0,0,0,0,0),False,user1)		
wobj8	Wobj	((0,0,0,0,0,0),False,user1)		
wobj9	Wobj	((0,0,0,0,0,0),False,user1)		
wobj10	Wobj	((0,0,0,0,0,0),False,user1)		
wobj11	Wobj	((0,0,0,0,0,0),False,user1)		

In the Gorilla Studio robot control system, 16 Work Object Coordinate Systems are pre-defined, of which wobj1 cannot be modified and coincides with the User Coordinate System. For coordinate systems other than wobj1, clicking the corresponding jump button  allows direct access to the calibration interface.

Note:

The modification of a work object definition can only be performed through the coordinate system calibration interface. For detailed steps, please refer to the "Calibrating Work Object Coordinate System" section. Wobj-type variables can only be viewed in the variable table interface; they cannot be created or modified.

Warning

Please do not attempt to directly modify Wobj-type variables in the program editing area, as this may cause unpredictable errors. If modifications are needed, please perform them through the calibration interface. For details, refer to "Calibrating Work Object Coordinate System."

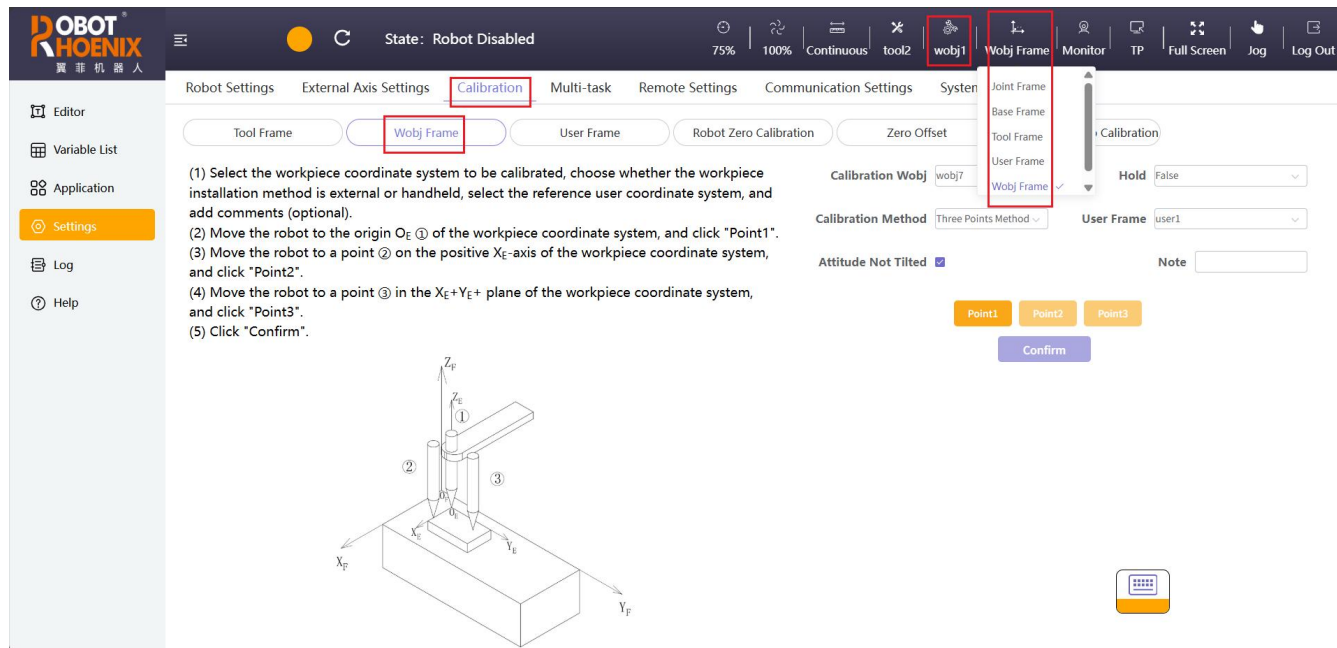
Calibrating Work Object Coordinate System

For detailed steps and methods for calibrating the Work Object Coordinate System, please refer to Section 2.4.2 Work Object Coordinate System Calibration in this manual.

3.3.3. Using Work Objects

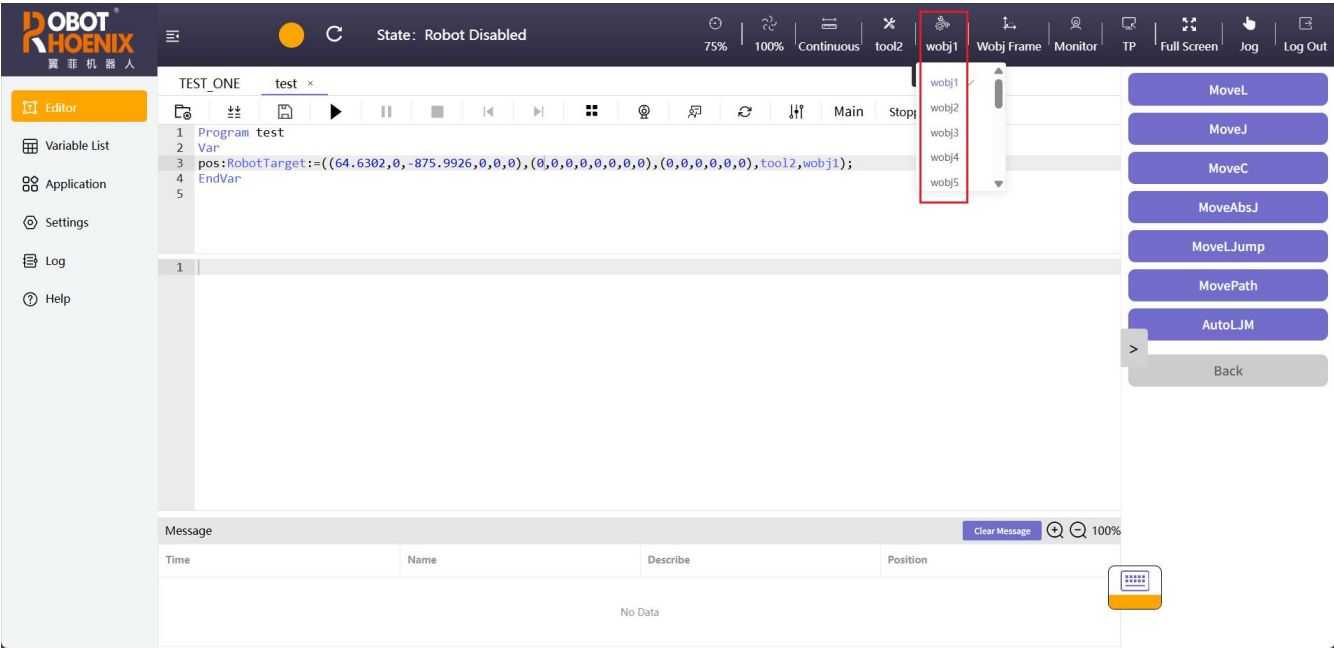
Using Work Objects in Jog Mode

To perform jog operations under a specific Work Object Coordinate System, click the Jog Coordinate System button () in the top toolbar, and select Work Object Coordinate System from the dropdown menu. Then click the Work Object Coordinate System button () to expand the dropdown list and select the desired work object. See the figure below:



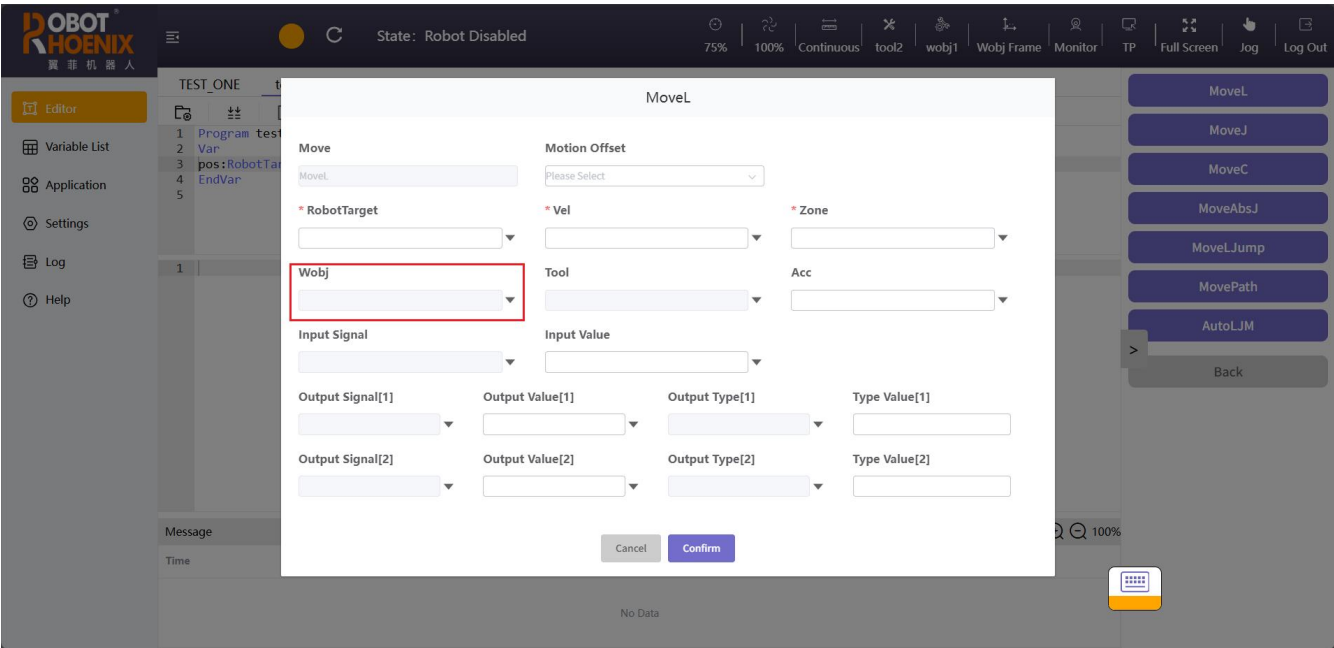
Use during teaching points

To use a specified workpiece when teaching local or global position variables, click the Work Object Coordinate System button () on the toolbar at the top of the interface, expand the dropdown menu, and select the desired workpiece. As shown in the figure below:

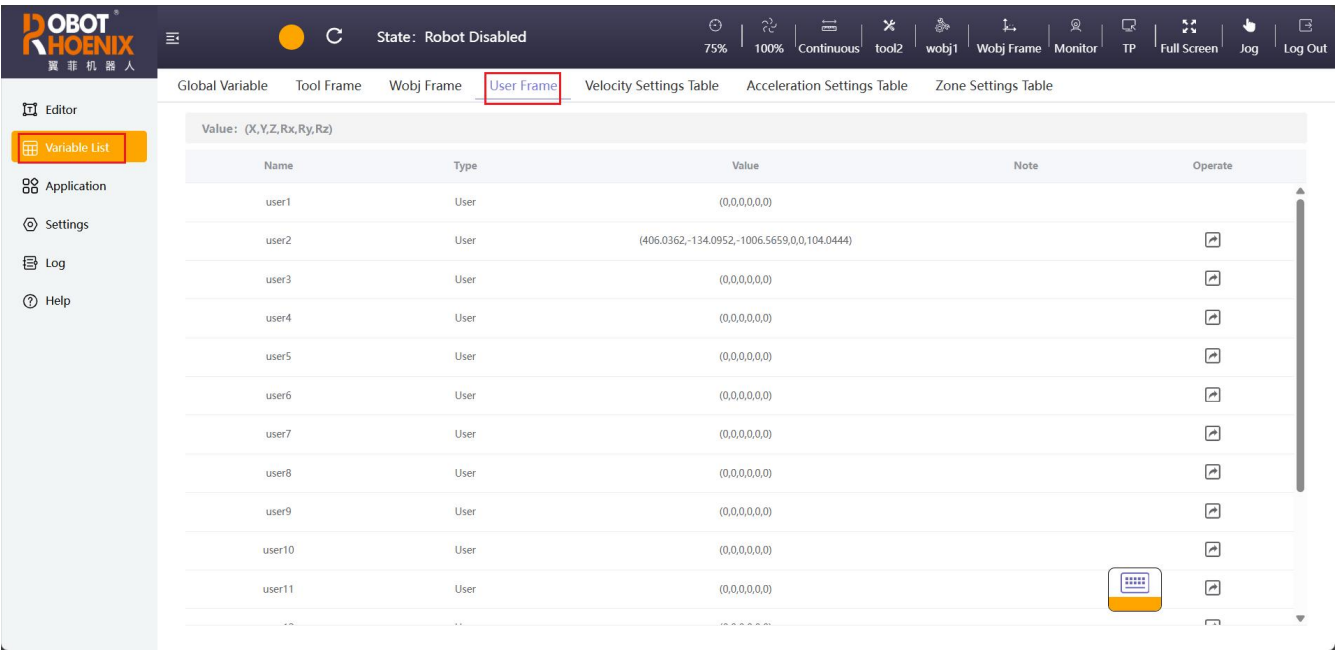


Use in the program

Using a specific work object in the program is straightforward—simply select the target work object in the “Wobj” parameter of the motion instruction. As shown below:



3.4. User Coordinate System



Definition

When defining a Work Object Coordinate System, it serves as a reference frame and is not used independently.

Explanation

In the Gorilla Studio robot control system, 16 User Coordinate Systems are predefined, among which user1 cannot be modified. For all other user coordinate systems, clicking the corresponding jump button  allows direct access to the calibration interface.

Internal format

Internal format of User-type variables: (X, Y, Z, Rx, Ry, Rz)

Note:

Modifications to a User definition can only be performed through the coordinate system calibration interface. For detailed steps, please refer to the “User Coordinate System Calibration” section. In the variable table interface, User-type variables can only be viewed; they cannot be created or modified.

3.5. Speed Setting Table

Definition

The Vel-type variable contains five member variables: Cartesian space speed percentage, joint space speed percentage, Cartesian space rotation speed percentage, MoveLJump upward segment speed percentage, and MoveLJump downward segment speed percentage.

Explanation

In the Gorilla Studio robot control system, eight commonly used velocity variables are pre-defined, as shown in the figure below:

The screenshot shows the ROBOT PHOENIX software interface. The top bar includes the logo, a state indicator 'State: Robot Disabled', and various control icons. The left sidebar contains navigation options like Editor, Variable List, Application, Settings, Log, and Help. The main area displays the 'Velocity Settings Table' with a red box highlighting its title. Above the table, a value string is shown: 'Value: (Velocity Percentage in Cartesian, Velocity Percentage in Joint, Rotation Velocity Percentage in Cartesian, Rising Velocity, Falling Velocity)'. The table itself has five columns: Name, Type, Value, Note, and Operate. It lists eight pre-defined velocity variables (vel1 to vel8) with their respective types and values.

Name	Type	Value	Note	Operate
vel1	Vel	(30,30,30,30,30)		
vel2	Vel	(20,20,20,20,20)		
vel3	Vel	(30,30,30,30,30)		
vel4	Vel	(40,40,40,40,40)		
vel5	Vel	(50,50,50,50,50)		
vel6	Vel	(60,60,60,60,60)		
vel7	Vel	(70,70,70,70,70)		
vel8	Vel	(100,100,100,100,100)		

Internal format

Internal format of Vel-type variables: (tcp, joint, rotation, jumpUp, jumpDown)

- tcp

Cartesian Space Speed Percentage

Used to specify the motion speed for Cartesian space movement commands. Applicable to MoveL, MoveC, MoveLJump, and MovePath instructions (for MoveLJump, this parameter only affects the horizontal segment). Allowed range: 1% to 100%.

- joint

Joint Space Speed Percentage

Used to specify the motion speed for joint space movement commands. Applicable to MoveAbsJ and MoveJ instructions. Allowed range: 1% to 100%.

- rotation

Cartesian Space Rotation Speed Percentage

Used to specify the rotational speed for Cartesian space movement commands. Applicable to MoveL, MoveC, MoveLJump, and MovePath instructions. Allowed range: 1% to 100%.

- jumpUp

MoveLJump Upward Segment Speed Percentage

Used to specify the speed of the upward segment for the MoveLJump command. Applicable to MoveLJump instructions. Allowed range: 1% to 100%.

- jumpDown

MoveLJump Downward Segment Speed Percentage

Used to specify the speed of the downward segment for the MoveLJump command. Applicable to MoveLJump instructions. Allowed range: 1% to 100%.

Edit and Annotation

To modify and annotate system-predefined speed variables, use the Velocity Setting Table.

As shown in the figure below, navigate to “Variable Table → Velocity Setting Table” from the left-hand menu to enter the Velocity Setting Table interface. Click the operation button () on the row of the speed variable you want to edit to modify or add annotations. After making the changes, click OK to save.

3.6. Acceleration Setting Table

Definition

Acc-type variables contain five member parameters: Cartesian space acceleration percentage, joint space acceleration percentage, Cartesian space rotational acceleration percentage, MoveLJump ascent-phase acceleration percentage, and MoveLJump descent-phase acceleration percentage.

Explanation

In the Gorilla Studio robot control system, eight commonly used acceleration variables are pre-defined, as shown in the figure below:

Value: (Acceleration Percentage in Cartesian,Acceleration Percentage in Joint,Rotation Acceleration Percentage in Cartesian,Rising Acceleration,Falling Acceleration)				
Name	Type	Value	Note	Operate
acc1	Acc	(10,10,10,10,10)		
acc2	Acc	(20,20,20,20,20)		
acc3	Acc	(30,30,30,30,30)		
acc4	Acc	(40,40,40,40,40)		
acc5	Acc	(50,50,50,50,50)		
acc6	Acc	(60,60,60,60,60)		
acc7	Acc	(70,70,70,70,70)		
acc8	Acc	(100,100,100,10,10)		

Internal format

Internal format of Acc-type variables: (tcp, joint, rotation, jumpUp, jumpDown)

- tcp

Cartesian space acceleration percentage

Used to specify the acceleration for Cartesian-space motion commands. This parameter applies to MoveL, MoveC, MoveLJump, and MovePath commands (for MoveLJump, it only takes effect on the horizontal segment). The valid value range is 1% to 100%.

- joint

Joint space acceleration percentage

Used to specify the acceleration for joint-space motion commands.

This parameter applies to MoveAbsJ and MoveJ commands.

The valid value range is 1% to 100%.

- rotation

Cartesian-space rotational acceleration percentage

Used to specify the rotational acceleration for Cartesian-space motion commands.

This parameter applies to MoveL, MoveC, MoveLJump, and MovePath commands.

The valid value range is 1% to 100%.

- jumpUp

MoveLJump ascent acceleration percentage

Used to specify the ascent acceleration for the MoveLJump command.

This parameter applies only to MoveLJump.

The valid value range is 1% to 100%.

- jumpDown

MoveLJump descent acceleration percentage

Used to specify the descent acceleration for the MoveLJump command.

This parameter applies only to MoveLJump.

The valid value range is 1% to 100%.

Modifications and Comments

Modifying and annotating the system's pre-defined acceleration variables can only be done through the Acceleration Setting Table.

As shown in the figure below, click “Variable Table” → “Acceleration Setting Table” in the left navigation bar to enter the Acceleration Setting Table interface. Click the operation button  in the row of the acceleration variable you want to modify to make changes and add annotations. Once finished, click OK to confirm.

3.7. Turn Area Dimension Setting Table

Overview

Zone variables are used to define the size of the turning area between two motion trajectories. Using a turning area can help avoid frequent starts and stops, thereby improving the robot’s motion cycle.

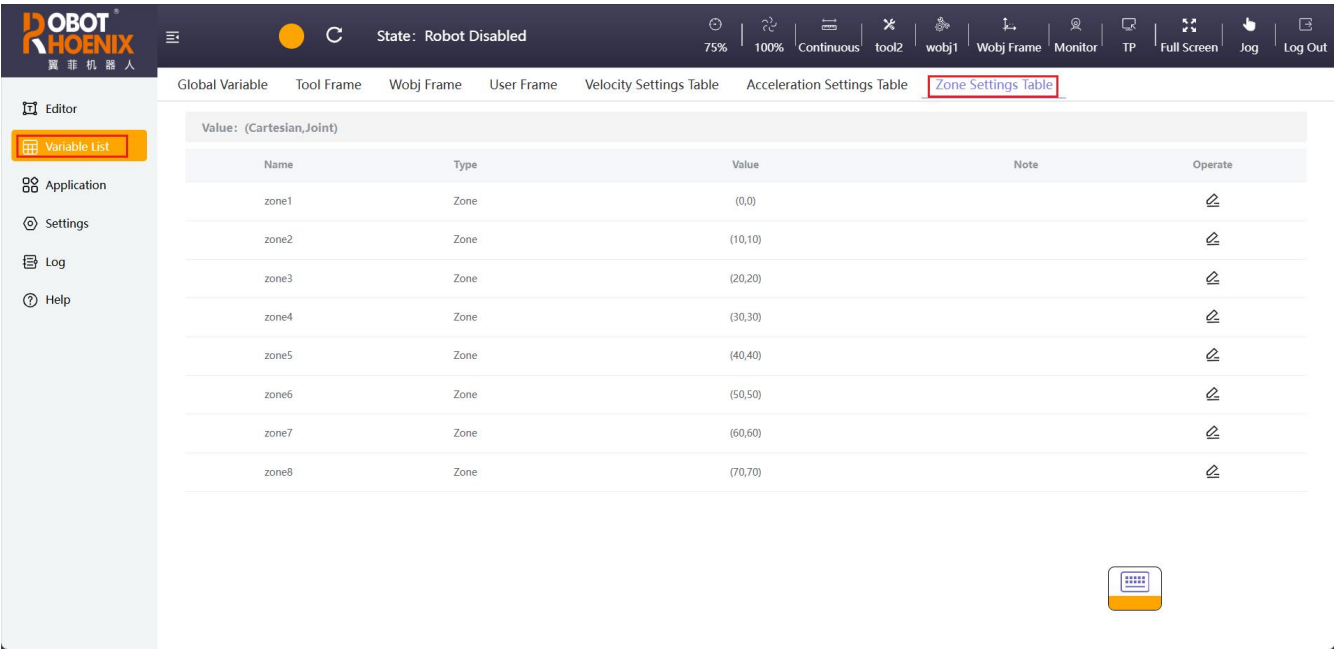
Definition

Joint-space trajectories and Cartesian-space trajectories use different parameters to define the turning area, so this variable includes two parts: distance and percent.

Explanation

The size of the defined turning area cannot exceed half of the path length. If it does, the system will automatically reduce the turning area to half of the total path length.

In the Gorilla Studio robot control system, eight commonly used turning area size variables are pre-defined, as shown in the figure below:



The screenshot shows the ROBOT PHOENIX software interface. The top bar includes the logo, a menu icon, a yellow circle, a refresh icon, and the text 'State: Robot Disabled'. The right side of the top bar contains various icons for zooming (75%, 100%), motion modes (Continuous, tool2, wobj1, Wobj Frame), and other functions (Monitor, TP, Full Screen, Jog, Log Out). Below the top bar is a navigation menu with 'Global Variable', 'Tool Frame', 'Wobj Frame', 'User Frame', 'Velocity Settings Table', 'Acceleration Settings Table', and 'Zone Settings Table' (which is highlighted with a red box). The main area displays the 'Zone Settings Table' with a header 'Value: (Cartesian,Joint)'. The table has five columns: Name, Type, Value, Note, and Operate. It lists eight pre-defined zones from zone1 to zone8, all of type 'Zone', with values ranging from (0,0) to (70,70). Each row has an 'Operate' icon. On the left side of the interface, there is a sidebar with icons for Editor, Variable List (highlighted in orange), Application, Settings, Log, and Help.

Name	Type	Value	Note	Operate
zone1	Zone	(0,0)		
zone2	Zone	(10,10)		
zone3	Zone	(20,20)		
zone4	Zone	(30,30)		
zone5	Zone	(40,40)		
zone6	Zone	(50,50)		
zone7	Zone	(60,60)		
zone8	Zone	(70,70)		

Internal format

Internal format of Zone-type variables: (distance, percent)

- distance

Cartesian-space turning area, unit: mm, value range: 0 to 1999.

It is used for MoveL, MoveC, MoveLJump, and MovePath commands. That is, when the robot reaches the specified distance from the target point, it begins to turn toward the next target point.

- percent

Joint-space turning area, unit: %, value range: 0% to 100%.

It is used for MoveAbsJ and MoveJ commands. The turning is based on the percentage of motion time; for example, 50% means the robot begins turning toward the next target point when half of the total motion time has elapsed.

Usage Limitations

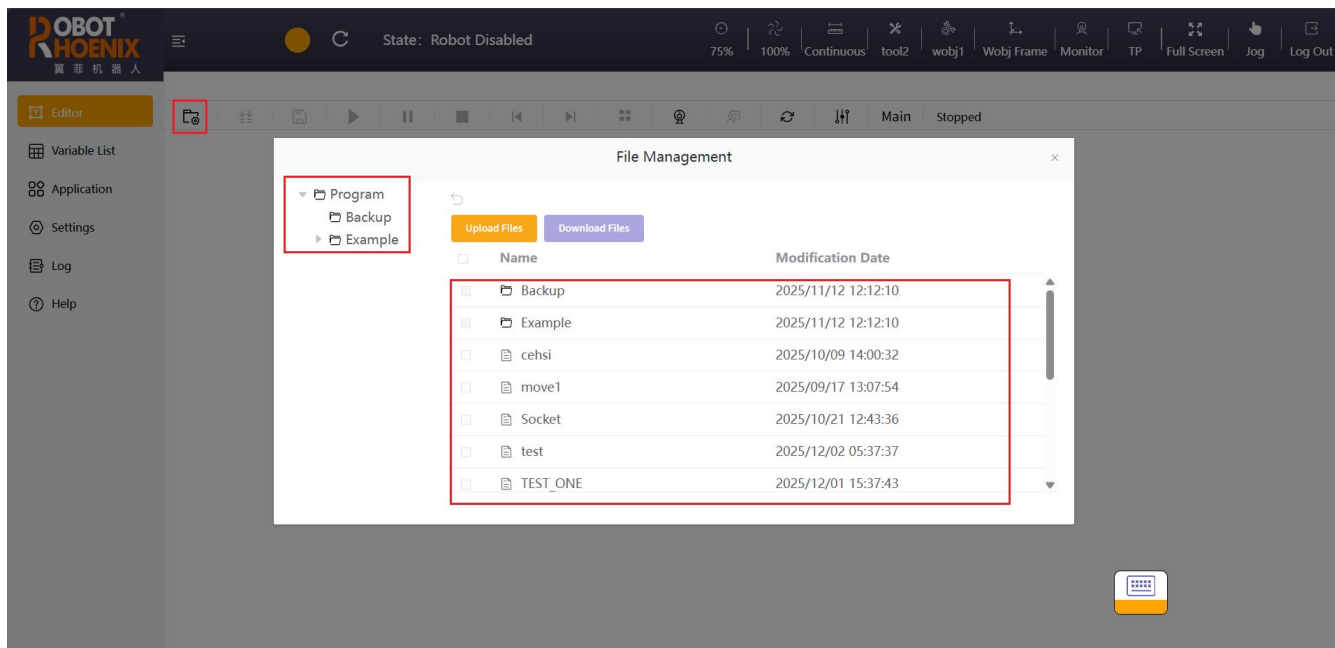
In certain specific situations, the turning area may be canceled. Possible reasons include:

- The “Input Signal” function is set in the motion command.
- The motion command is the last command in a sequence.
- There are non-motion commands between motion commands.
- Motion commands at the beginning and end of a For or While loop.
- The “External Axis” function is enabled.

4. Editor

4.1. Program File List

Click the “File Management” button () in the editing area to enter the File Management interface, as shown in the figure below:



The left side shows the list of program folders, and the right side shows the expanded sub-list of a folder, where:

- The “Backup” folder contains deleted program files. Programs in the “Backup” folder cannot be edited, run, or otherwise operated on. After copying them to another folder, they can be edited and executed.
- The “Example” folder is the sample program folder, containing sample programs such as “Math,” “Modbus,” “Movement,” “SocketClient,” “SocketServer,” and “StringParsing.” Sample programs opened directly cannot be edited or run; they must first be copied to another folder before editing or execution.

Note:

After a program is successfully compiled or while it is running, program files and folders cannot be operated on.

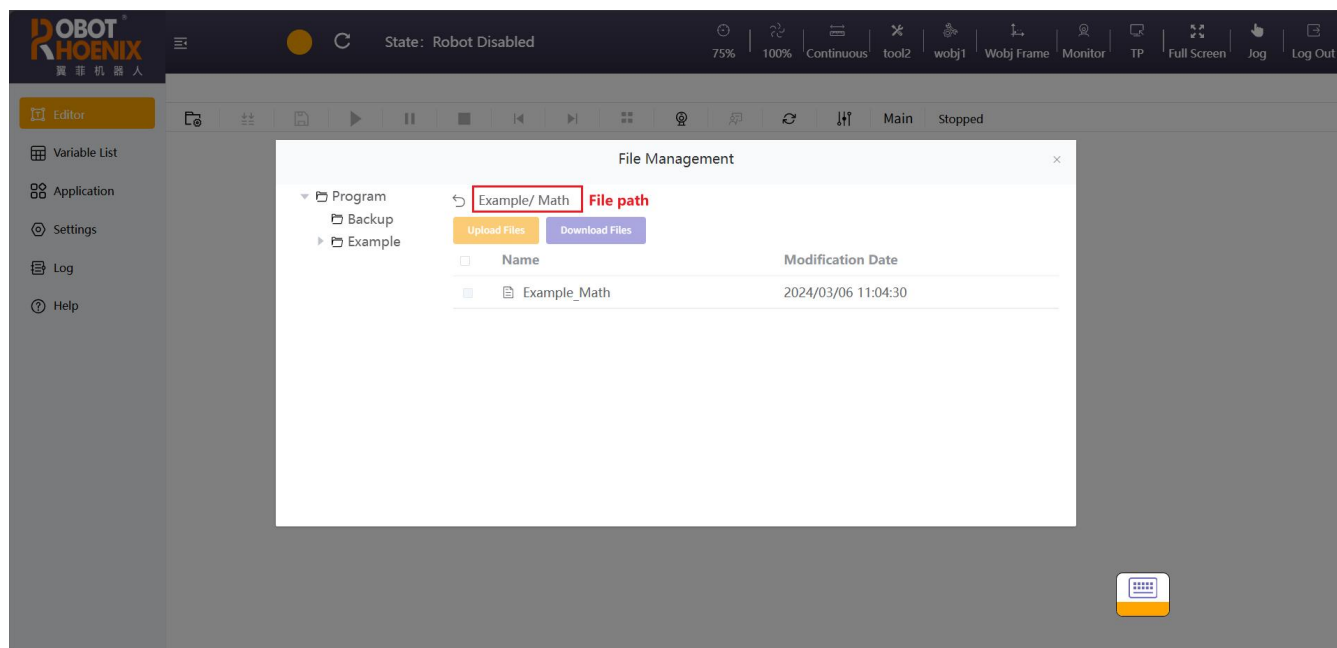
After a program is successfully compiled, files can be uploaded or downloaded; during program execution, files cannot be uploaded or downloaded.

Folder names and file names cannot contain spaces or the following special characters: \ / : * ? ' < > |

Folder icon  indicates a folder, file icon  indicates a program file.

Click the button  to return to the previous folder level.

Click a folder name in the file path to return to that folder, as shown in the figure below:



4.1.1. Folder Operations

Operation Method

- Create New Folder

- (1) Method 1: Right-click in the program folder list on the left and select Create New Folder.
- (2) Method 2: Right-click on the header (Name – Modified Time) of the sub-list on the right and select Create New Folder.
- (3) Method 3: Right-click in the sub-list area on the right and select Create New Folder.

- Copy Folder

(1) Method 1: Right-click the corresponding folder in the program folder list on the left and select Copy.

(2) Method 2: Right-click the corresponding folder in the sub-list on the right and select Copy.

- Paste Folder

(1) Method 1: Right-click in the program folder list on the left and select Paste.

(2) Method 2: Right-click on the header (Name – Modified Time) of the sub-list on the right and select Paste.

(3) Method 3: Right-click in the sub-list area on the right and select Paste.

- Delete Folder

(1) Method 1: Right-click the corresponding folder in the program folder list on the left and select Delete.

(2) Method 2: Right-click the corresponding folder in the sub-list on the right and select Delete.

- Rename Folder

(1) Method 1: Right-click the corresponding folder in the program folder list on the left and select Rename.

(2) Method 2: Right-click the corresponding folder in the sub-list on the right and select Rename.

Note:

The “Program” folder cannot be copied, deleted, or renamed.

The “Backup” folder cannot be operated on, but its subfolders and program files can be copied and deleted.

The “Example” folder cannot be operated on, but its subfolders and program files can be copied.

4.1.2. Program File Operations

Operation Method

- Create New Program File

(1) Method 1: Right-click in the program folder list on the left and select Create New Program.

(2) Method 2: Right-click on the header (Name – Modified Time) of the sub-list on the right and select Create New Program.

(3) Method 3: Right-click in the sub-list area on the right and select Create New Program.

- Copy New Program File

Right-click the corresponding file in the sub-list on the right and select Copy.

- Paste New Program File

(1) Method 1: Right-click on the header (Name – Modified Time) of the sub-list on the right and select Paste.

(2) Method 2: Right-click in the sub-list area on the right and select Paste.

- Delete New Program File

Right-click the corresponding file in the sub-list on the right and select Delete.

- Rename New Program File

Right-click the corresponding file in the sub-list on the right and select Rename.

- Upload New Program File

Click the “Upload File” button in the sub-list area on the right.

If a program file with the same name exists, you can choose:

Replace Program (only replaces the program, keeping local variables),

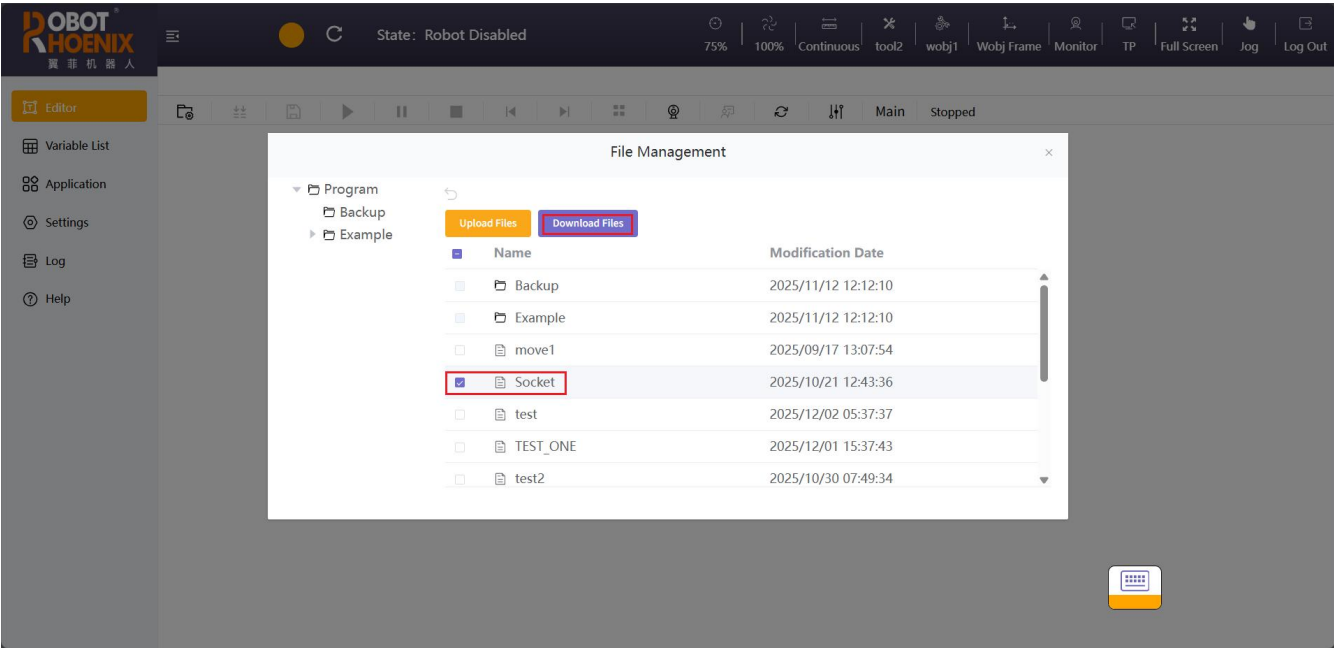
Replace All (replaces both local variables and the entire program), or

Do Not Copy (the file in the folder will remain unchanged).

- Download New Program File

In the sub-list area on the right, check the program files you want to download (multiple files can be selected) or check “Name” to select all program files in the current folder, then click the “Download File” button.

As shown in the figure below:

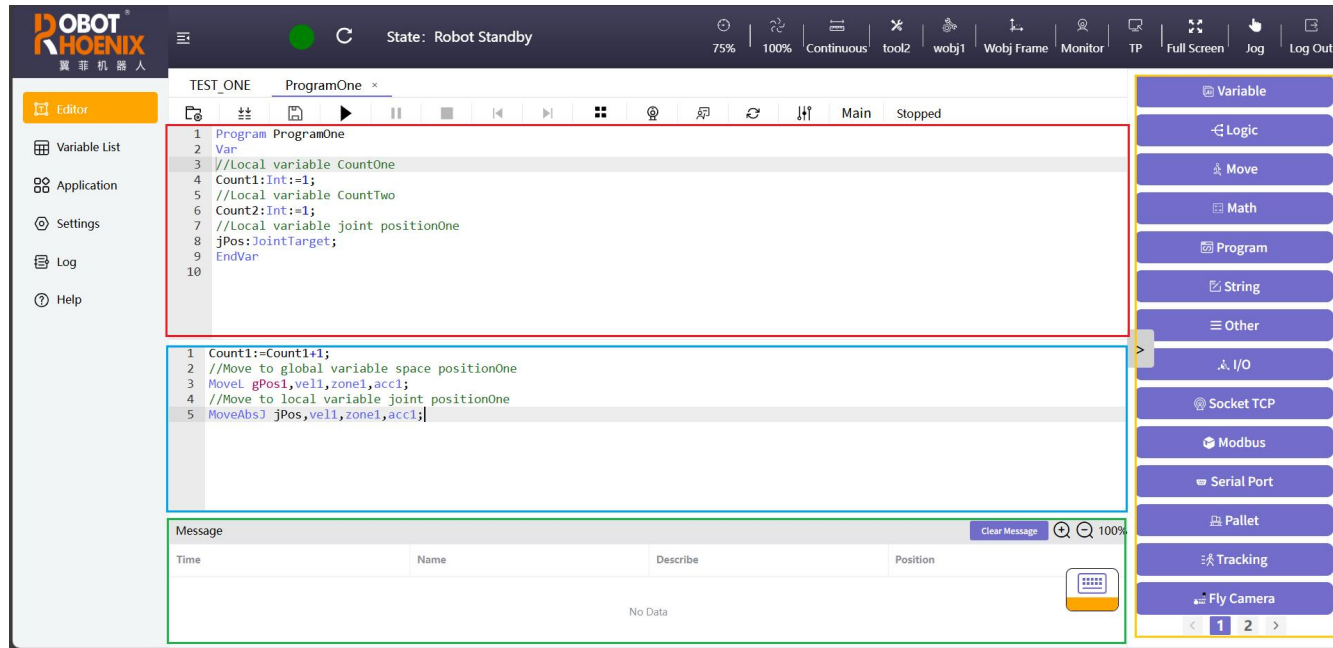


4.2. Program File Editing

The Gorilla Studio robot control system supports syntax hints, including variable suggestions, keyword suggestions, member variable hints, and statement auto-completion.

4.2.1. Example

To explain the features of a Gorilla Studio robot control system program, let's first look at a simple program to understand its basic structure and format, as shown in the figure below:



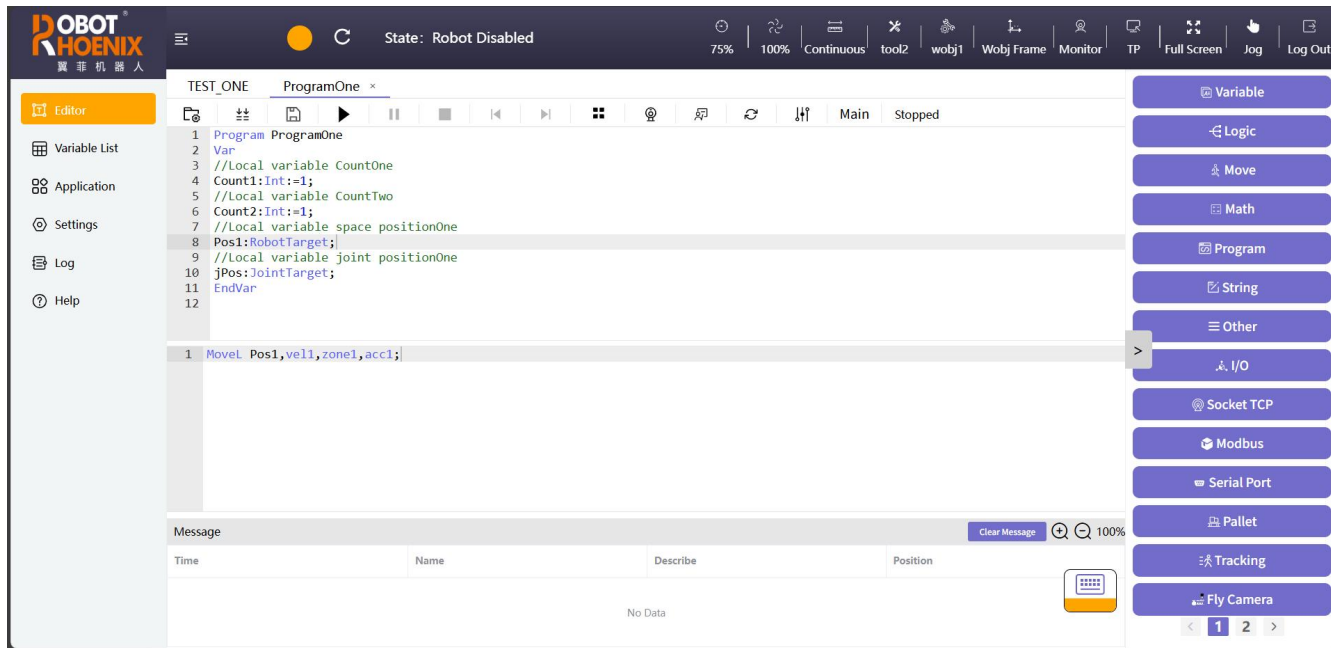
Herein:

- The entire program is divided into two main parts: the Variable Declaration Area (red box) and the Program Editing Area (blue box).
- Compilation results are displayed in the Problem Message Window at the bottom (green box).
- Variables can be quickly declared and instructions inserted via the Shortcut Instruction Module on the right (orange box).
- “Program” and “Program1” are the opened program files, with “Program1” being the currently edited program file.
- After “Program” comes the program name. “Var” marks the start of variable declarations, and “EndVar” marks the end. Variables must be declared between the “Var” and “EndVar” lines.
- The “Program”, “Var”, and “EndVar” lines cannot be modified or have line operations applied.
- The format for variable definitions is: VariableName:VariableType:=InitialValue;
- Statements end with a semicolon ;.
- Content after // is a comment and does not participate in program compilation.
- Global variables are highlighted in red.

4.2.2. Basic Operations

- Open File

In the “File Management” interface, double-click a program file to open it and enter the Program Editing interface. The status bar will display the name of the currently opened file, the run mode, and the run status, as shown in the figure below:



- Save File

Click the Save button () in the status bar to save the program.

- Compile File

Click the Compile button () in the status bar to compile the program. Compilation information can be viewed in the “Problem Message” panel below.

- Close File

Click the icon  next to the program file name to close the file.

4.2.3. Local Variable Declaration

Variables support array format

The entire program is divided into the Variable Declaration Area and the Program Editing Area. Unlike the global variables in the “Variable Table,” variables defined in the Variable Declaration Area are local variables and are effective only within the current program.

The currently supported variable types include:

Type	Variable	Description
Basic Type	Int	Integer: The allowed value range is -2147483647 to 2147483647.
	Double	Double-precision float: The allowed value range is -9999999999999999 to 9999999999999999.
	Float	Single-precision float type, with an allowable value range from -999999999999.999 to 999999999999.999.
	Bool	Boolean type, with values of either True or False.
	String	String type, allowing up to 1000 characters.
	Word	Unsigned short integer, with an allowable range from 0 to 65535.
Complex Type	Vel	Speed, with an internal format of (tcp, joint, rotation, jumpUp, jumpDown). tcp (%): Cartesian space speed percentage, with an allowable range from 1 to 100. joint (%): Joint space speed percentage, with an allowable range from 1 to 100. rotation (%): Cartesian rotation speed percentage, with an allowable range from 1 to 100. jumpUp (%): Ascending segment speed. The speed percentage for the ascending segment of the MoveLJump command, with an allowable range of 1% to 100%. jumpDown (%): Descending segment speed. The speed percentage for the descending segment of the MoveLJump command, with an allowable range of 1% to 100%.

	Acc	Acceleration, with an internal format of (tcp, joint, rotation, jumpUp, jumpDown).. tcp (%): Cartesian space acceleration percentage, with an allowable range from 1 to 100. joint (%): Joint space acceleration percentage, with an allowable range from 1 to 100. rotation (%): Cartesian rotation acceleration percentage, with an allowable range from 1 to 100. jumpUp (%): Ascending segment acceleration. The acceleration percentage for the ascending segment of the MoveLJump command, with an allowable range of 1% to 100%. jumpDown (%): Descending segment acceleration. The acceleration percentage for the descending segment of the MoveLJump command, with an allowable range of 1% to 100%.
	Zone	Zone, with an internal format of (distance, percent). distance (mm): Cartesian space zone distance, with an allowable range from 0 to 1999. percent (%): Joint space zone percentage, with an allowable range from 0 to 100.
	Pose	Pose, with an internal format of (X, Y, Z, Rx, Ry, Rz). X (mm) : The allowable value range is -9999 to 9999. Y (mm) : The allowable value range is -9999 to 9999. Z (mm) : The allowable value range is -9999 to 9999. Rx (°) : The allowable value range is -9999 to 9999. Ry (°) : The allowable value range is -9999 to 9999. Rz (°) : The allowable value range is -9999 to 9999.

	AxisPos	Axis Position, internal format: (J1, J2, J3, J4, J5, J6). J1 (°) : 1 Axis Position, The allowable value range is -180 to 180. J2 (°) : 2 Axis Position, The allowable value range is -180 to 180. J3 (°) : 3 Axis Position, The allowable value range is -9999 to 9999. J4 (°) : 4 Axis Position, The allowable value range is -999999999 to 999999999 J5 (°) : 5 Axis Position, The allowable value range is -360 to 360 J6 (°) : 6 Axis Position, The allowable value range is -360 to 360
	RobotTarget	● Internal format: (pose, confData, eAxisPos, tool, wobj) ● pose (Target Pose) : ● X (mm) : The allowable value range is -9999 to 9999. ● Y (mm) : The allowable value range is -9999 to 9999. ● Z (mm) : The allowable value range is -9999 to 9999. ● Rx (°) : The allowable value range is -9999 to 9999. ● Ry (°) : The allowable value range is -9999 to 9999. ● Rz (°) : The allowable value range is -9999 to 9999. ● confData (Configuration) : ● 1 Axis Rotation Quadrant: The allowable value range is -4 to 3. ● 2 Axis Rotation Quadrant: The allowable value range is -4 to 3. ● 3 Axis Rotation Quadrant: The allowable value range is -300 to 300.

		● 4 Axis Rotation Quadrant: The allowable value range is -300 to 300. ● 5 Axis Rotation Quadrant: The allowable value range is -4 to 3. ● 6 Axis Rotation Quadrant: The allowable value range is -4 to 3. ● 7 Axis Rotation Quadrant: The allowable value range is -4 to 3. ● Pose Type: The allowable value range is 0 to 8. ● eAxisPos (External Axis Position) : ● E1(mm): External Axis 1 Axis Position, The allowable value range is -9999 to 9999. ● E2(mm): External Axis 2 Axis Position, The allowable value range is -9999 to 9999. ● E3(mm): External Axis 3 Axis Position, The allowable value range is -9999 to 9999. ● E4(mm): External Axis 4 Axis Position, The allowable value range is -9999 to 9999. ● E5(mm): External Axis 5 Axis Position, The allowable value range is -9999 to 9999. ● E6(mm): External Axis 6 Axis Position, The allowable value range is -9999 to 9999. ● tool(Tool Coordinate System): The allowable value range is tool1 to tool16. ● wobj (Work object coordinate system) : The allowable value range is wobj1 to wobj16
	JointTarget	Internal format: (jointPos, eAxisPos) ● jointPos (Axis Position) : J1 (°) : 1 Axis Position, The allowable value range is -180 to

		180. J2 (°) : 2 Axis Position, The allowable value range is -180 to 180. J3 (°) : 3 Axis Position, The allowable value range is -9999 to 9999. J4 (°) : 4 Axis Position, The allowable value range is -9999999999 to 9999999999 J5 (°) : 5 Axis Position, The allowable value range is -360 to 360 J6 (°) : 6 Axis Position, The allowable value range is -360 to 360 ● eAxisPos (External Axis Position) : E1(mm): External Axis 1 Axis Position, The allowable value range is -9999 to 9999. E2(mm): External Axis 2 Axis Position, The allowable value range is -9999 to 9999. E3(mm): External Axis 3 Axis Position, The allowable value range is -9999 to 9999. E4(mm): External Axis 4 Axis Position, The allowable value range is -9999 to 9999. E5(mm): External Axis 5 Axis Position, The allowable value range is -9999 to 9999. E6(mm): External Axis 6 Axis Position, The allowable value range is -9999 to 9999.
	JumpData	Internal format: (upDis , upPer , downDis , downPer , rotUpDis, rotDownDis) upDis (Ascending Distance) (mm): The ascending distance for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable

		range is -999 to 999. upPer (Ascending Zone Percentage) (%): The percentage of the ascending zone in a door-shaped motion. The allowable range is 0 to 100. downDis (Descending Distance) (mm): The descending distance for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999. downPer (Descending Zone Percentage) (%): The percentage of the descending zone in a door-shaped motion. The allowable range is 0 to 100. rotUpDis (Pose Start Height) (mm): The pose start height for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999. rotDownDis (Pose Stop Height) (mm): The pose stop height for a door-shaped motion. + indicates upward along the Z-axis of the specified Work Object coordinate system in the command, and - indicates downward along the Z-axis. The allowable range is -999 to 999.
--	--	---

Operation Method

● Manual Input

Place the cursor between the “Var” and “EndVar” lines, enter the variable name and type. Each statement ends with a semicolon ;, and multiple statements can be on a single line.

The format is:

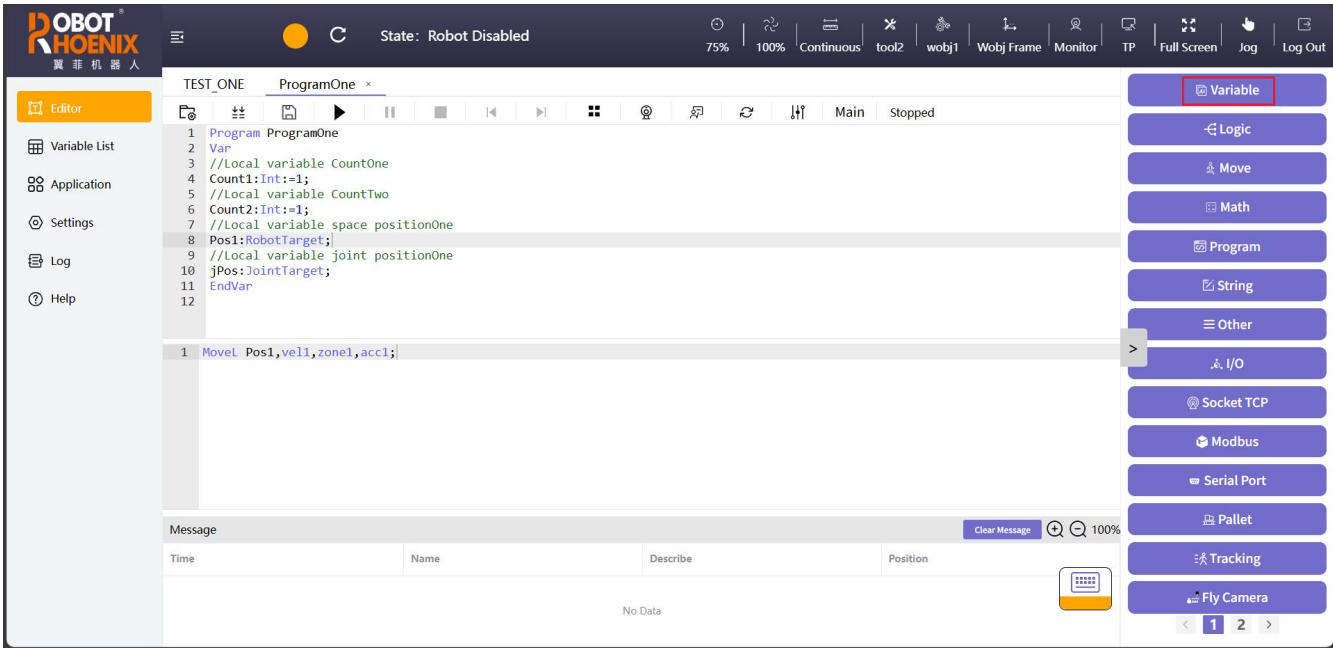
VariableName : VariableType := InitialValue;

or VariableName : VariableType;

● **Shortcut Commands**

Place the cursor between the “Var” and “EndVar” lines, expand the Shortcut Commands module on the right, click “Variable,” select the variable type, enter the variable name, dimensions, and initial value, and click OK.

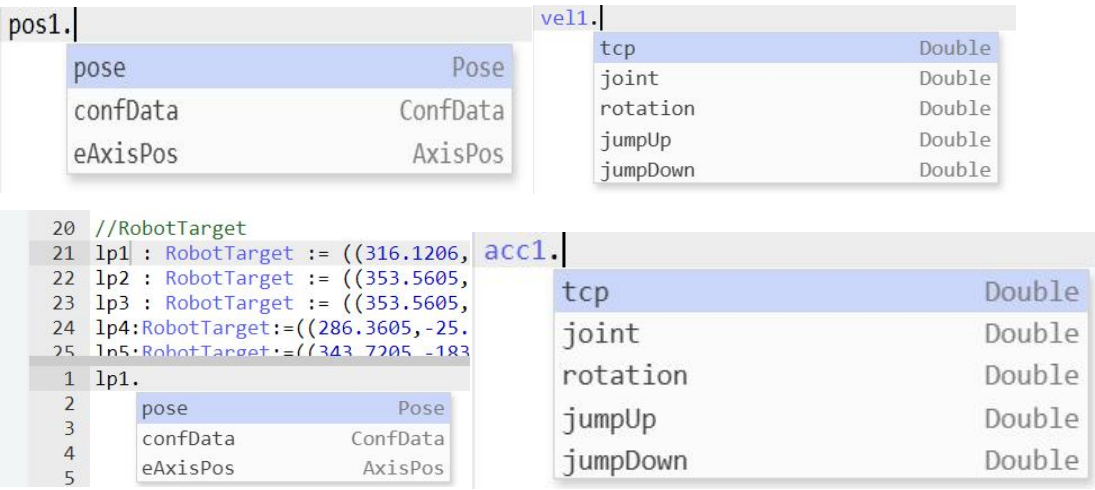
As shown in the figure below:



Note:

The “Program,” “Var,” and “EndVar” lines cannot be modified.

Variables of types JointTarget, RobotTarget, JumpData, Zone, Vel, Acc, Pose, and AxisPos in both global and local variables support the “.” (dot) operator, as shown in the figure below:



4.2.4. Program Editing

4.2.4.1. Operation Method

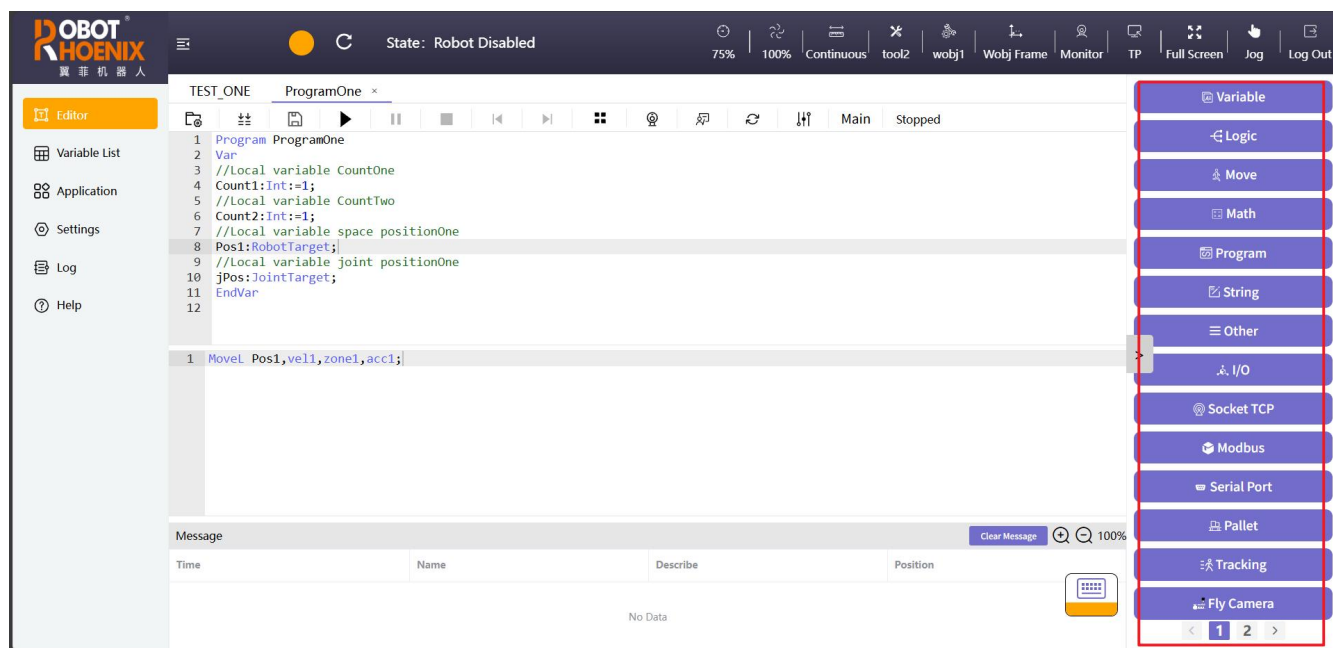
- Manual Input

Place the cursor in the Program Editing area and manually enter the program content. Each statement ends with a semicolon “;”, and multiple statements can be on a single line.

- Shortcut Commands

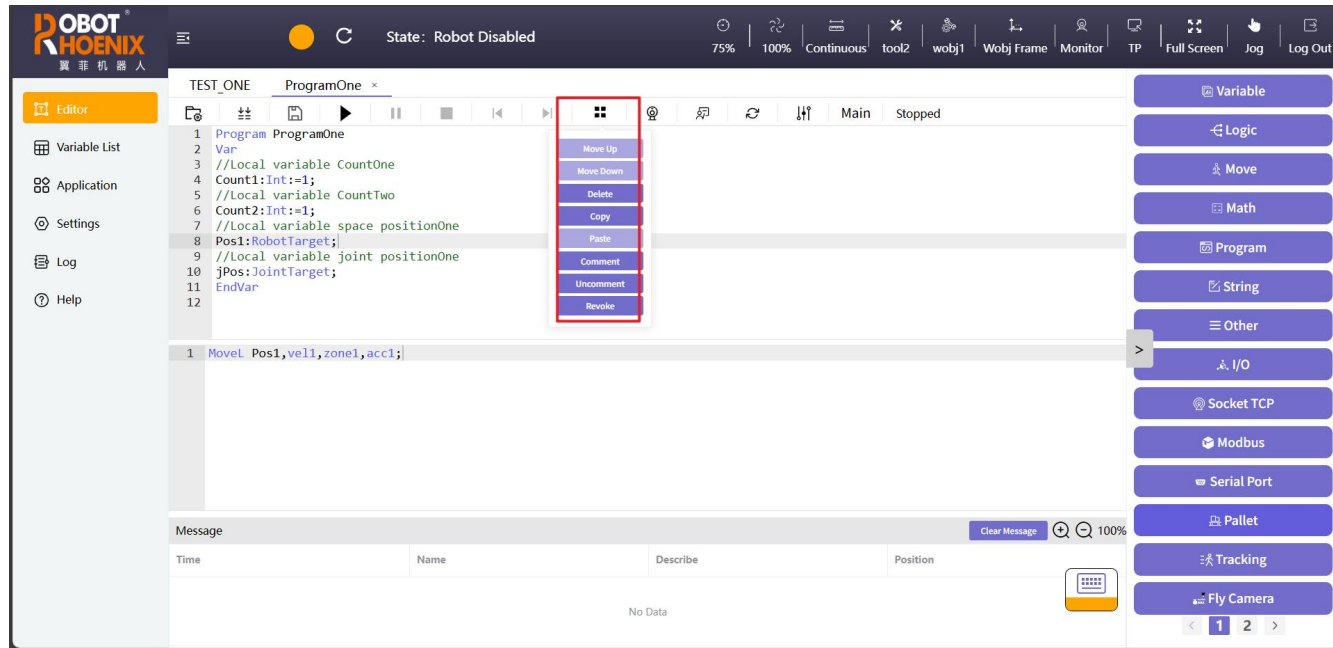
Place the cursor in the Program Editing area, expand the Shortcut Commands module on the right, click the desired command, fill in the required information, and click OK.

As shown in the figure below:



4.2.4.2. Operation Menu

Click the operation button () to open the Operation Menu dropdown. Click again to collapse it. The Operation Menu includes the following functions: Move Line Up, Move Line Down, Delete Line, Copy, Paste, Comment, Uncomment, and Undo, as shown in the figure below:



- Move Line Up

Place the cursor on a line and click Move Line Up to move the line up by one. The first line cannot be moved up.

- Move Line Down

Place the cursor on a line and click Move Line Down to move the line down by one. The last line cannot be moved down.

- Delete Line

Place the cursor on a line and click Delete Line to remove that line.

- Copy

Place the cursor on a line and click Copy to copy the entire line.

Select specific content and click Copy to copy only the selected content.

- Paste

Place the cursor at a position in the Variable Declaration Area and click Paste to insert only the content previously copied from the Variable Declaration Area.

Similarly, place the cursor in the Program Editing Area and click Paste to insert only the content previously copied from the Program Editing Area.

- Comment

Place the cursor on a line or select part of a line and click Comment to comment out the entire line.

If multiple lines are selected, all selected lines will be commented.

A single line can be commented multiple times.

You can also use the shortcut Ctrl + / to comment.

- **Uncomment**

Place the cursor on a line that is not commented or select part of an uncommented line and click Uncomment to comment the line.

Place the cursor on a commented line or select part of a commented line and click Uncomment to remove one layer of comment from the line.

If multiple selected lines are all commented, clicking Uncomment will remove one layer of comment from all selected lines.

If multiple selected lines are all uncommented, clicking Uncomment will add one layer of comment to all selected lines.

If multiple selected lines include both commented and uncommented lines, clicking Uncomment will add one layer of comment to all selected lines.

You can also use the shortcut Ctrl + / to uncomment.

- **Undo**

Place the cursor at a position in the Variable Declaration Area and click Undo to revert the last operation in the Variable Declaration Area.

Place the cursor at a position in the Program Editing Area and click Undo to revert the last operation in the Program Editing Area.

That is, undo operations in the Variable Declaration Area and the Program Editing Area are independent and do not affect each other.

Note:

The “Program,” “Var,” and “EndVar” lines do not support the above line operations.

4.2.4.3. Teach

JointTarget and RobotTarget type variables defined in the Variable Declaration Area can be taught to the current pose by clicking the Teach button ().

Note: For RobotTarget type variables, the tool and wobj parameters are taught as the tool number and work object number currently selected in the toolbar.

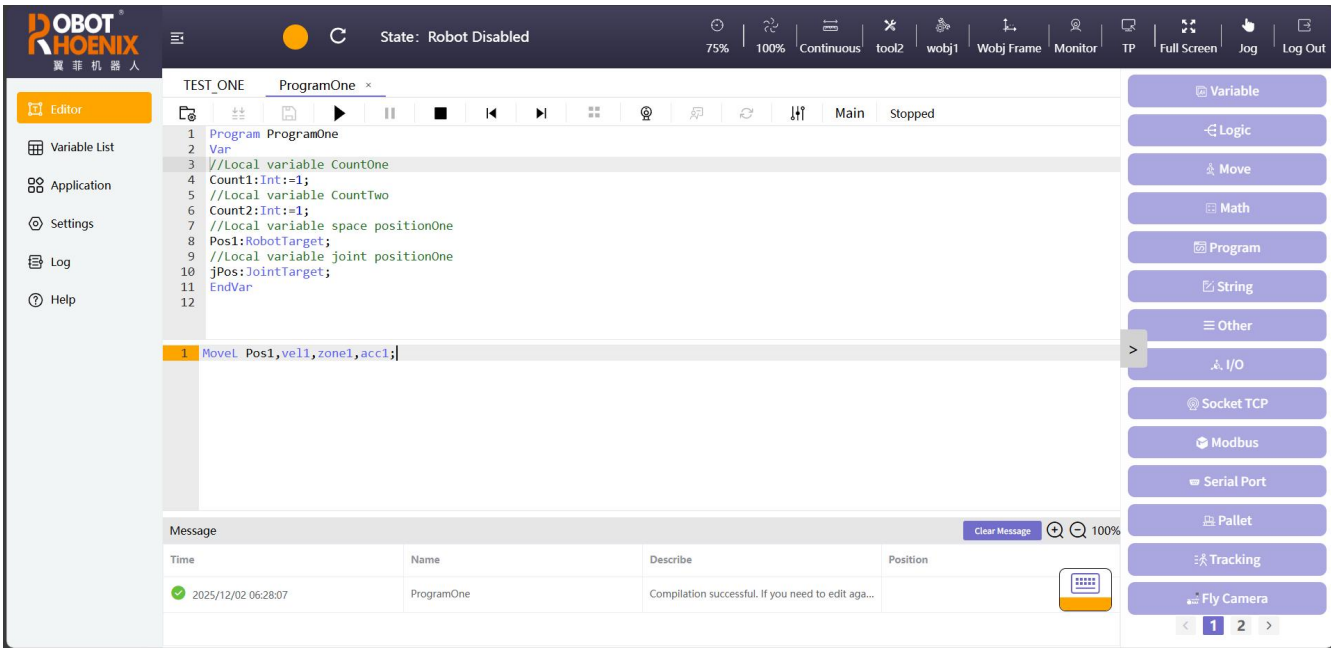
4.3. Save

After completing the program, click Save () to save it.

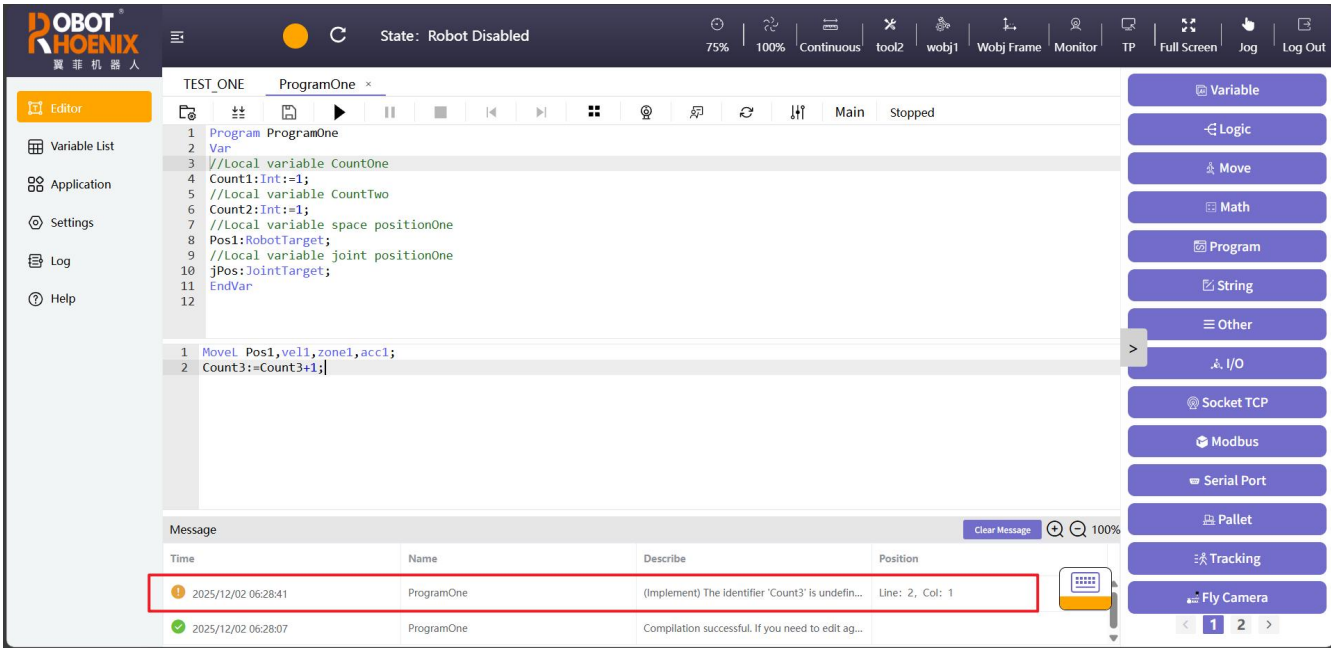
4.4. Compile

Click the Compile button () to compile the program. Compilation information is displayed in the Problem Message window.

- After successful compilation, the program switches to an executable state. The program run cursor () shows the corresponding line in the Program Editing area. At this time, program editing is disabled, and the Shortcut Commands insertion module is inactive. To edit the program again, click Stop (). With the robot enabled, click Start () or Step () to run the program, as shown in the figure below:



- If there are errors in the compilation, the Problem Message panel displays the location and details of the errors. The program remains in an editable state and must be corrected and compiled again, as shown in the figure below:



4.5. Debugging

After successful compilation, the program switches to an executable state and can be run when the robot is enabled.

4.5.1. Program Status

There are three program states: Stop, Run, and Pause.

● Stop



At this time, the program is in an editable state. You can modify the program content and switch the program loop mode.

● Run



At this time, the program is in a non-editable state. You cannot modify the program content or switch the program loop mode.

“A_Teach/Test2” is the file path of the program currently being executed.

● Pause



At this time, the program is not editable. You cannot modify the program content or change the program loop mode.

“A_Teach/Test2” is the file path of the program currently being executed.

4.5.2. Basic Operations

4.5.2.1. Start

Click “Start” (▶), and the program state switches to “Run.” The program becomes non-editable, and the run mode is automatic. After executing the line indicated by the program run cursor, the program continues to execute subsequent lines. The following explains three usage scenarios:

- Click “Start” with the program written but not yet compiled. In this case, clicking Start will first compile the program. If there are errors, they will be displayed in the message window, the program will not execute, and the program state remains Stop. If there are no errors, the program executes and the state switches to Run.
- Click “Start” with the program written and successfully compiled. In this case, clicking Start immediately executes the program, and the program state switches to Run.
- Click “Start” when the program is paused. In this case, clicking Start continues executing the paused instructions, and the program state switches to Run.

4.5.2.2. Pause

Explanation

Click “Pause” (), and the program state switches to Pause. The current program stops executing, and the program remains non-editable.

4.5.2.3. Stop

Explanation

Click “Stop” (), and the program state switches to Stop. The current program stops executing, and the program becomes editable.

4.5.2.4. Previous Step

Explanation

Click “Previous Step” (), and the program state switches to Run. The previous statement is executed, and the program becomes non-editable.

Note

“Previous Step” is only effective in step mode for motion instructions that have been executed using “Next Step.”

4.5.2.5. Next Step

Explanation

Click “Next Step” () , and the program state switches to Run. The next statement is executed, or a paused instruction continues execution, and the program becomes non-editable.

4.5.3. Loop Mode

The program loop mode includes Loop Run () and Single Run () . Click the icon to switch between the two modes:

- Loop Run: After executing the last line, the program continues running from the first line.
- Single Run: After executing the last line, the program stops running and switches to an editable state.

Note:

The program loop mode cannot be switched while the program is running or paused.

The loop mode applies to all programs simultaneously; changing the loop mode of one program will also change the loop mode of other programs.

4.5.4. Run Modes

The program run modes are Automatic Run and Step Run:

- Automatic Run: Click Start () , and after executing the line indicated by the program run cursor, the program continues to execute subsequent lines.
- Step Run: Click Next Step () , and after executing the line indicated by the program run cursor, the program stops.

Note:

In Automatic Run mode, the program executes automatically. Zone settings and motion look-ahead are effective.

In Step Run mode, zone settings and motion look-ahead are not effective.

In Step Run mode, clicking Next Step while the robot is moving will have no effect.

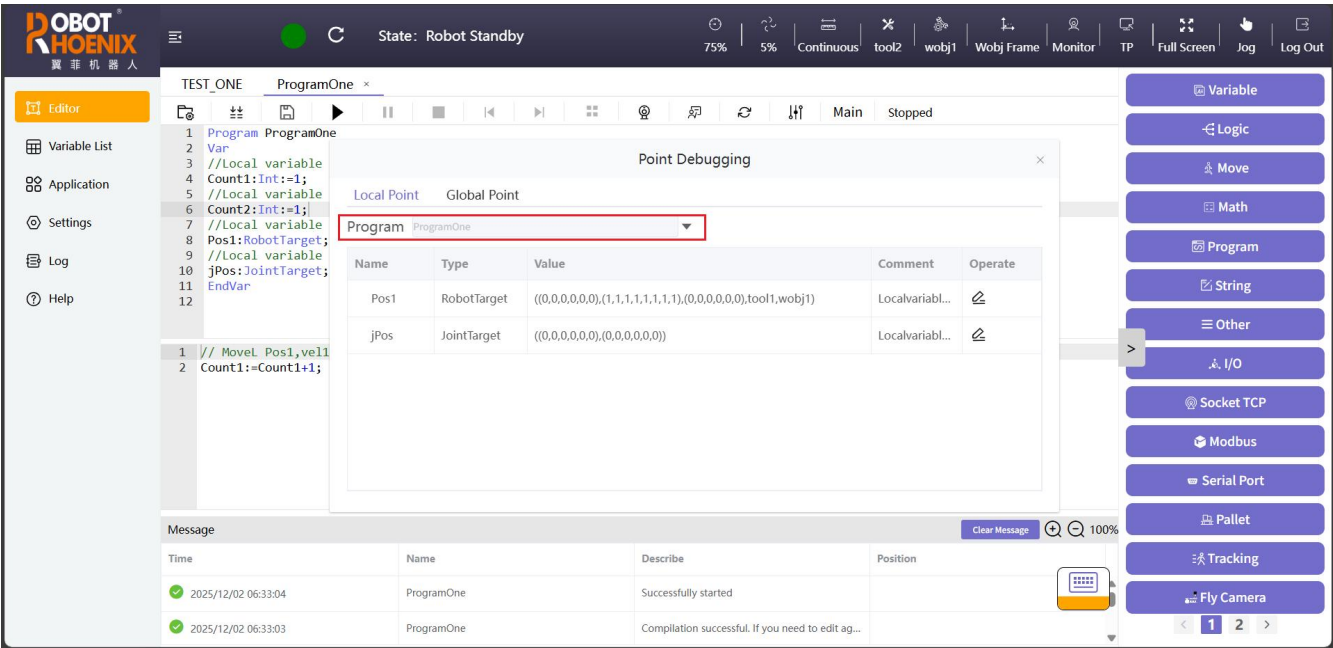
The run mode can be switched at any time during program execution.

In Automatic Run mode, clicking Pause (⏸) and then Next Step (⏭) switches the program to Step Run mode; after executing the line indicated by the program run cursor, execution stops.

In Step Run mode, clicking Pause (⏸) and then Start (▶) switches the program to Automatic Run mode; after executing the line indicated by the program run cursor, the program continues with subsequent lines.

4.5.5. Point Debugging

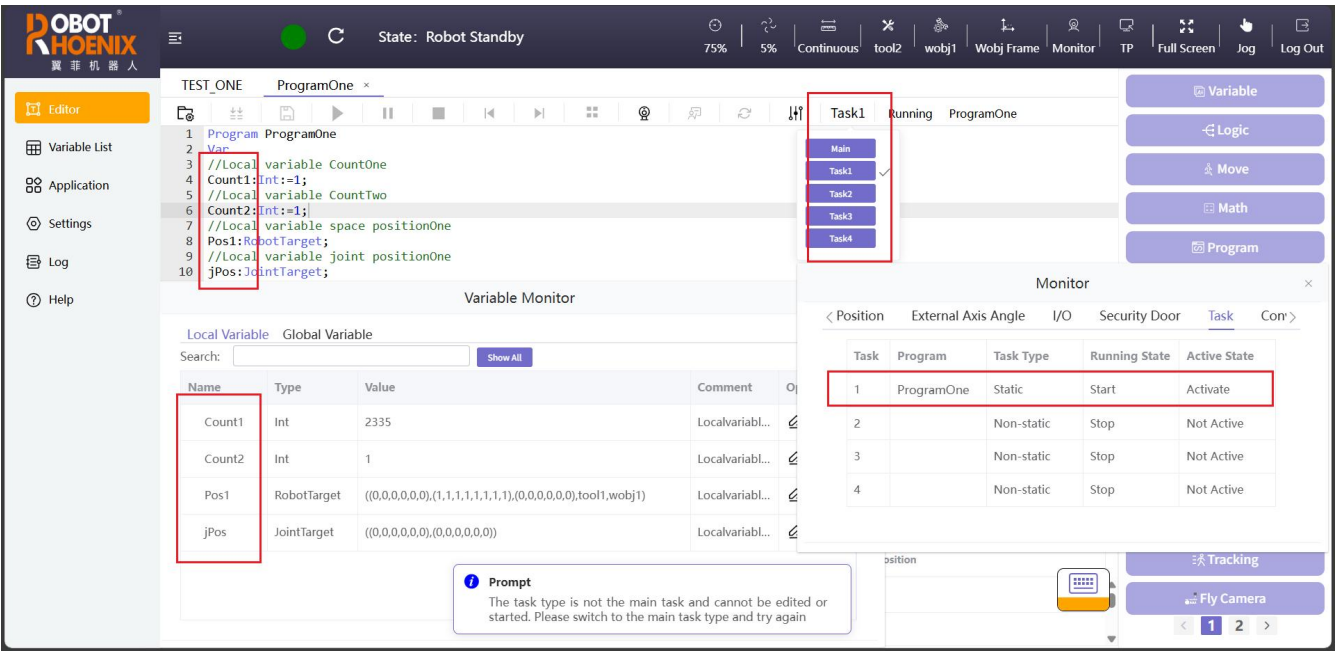
Click the Point Debugging (📍) button to modify the information of local points (RobotTarget and JointTarget variables) and global points (RobotTarget and JointTarget variables) in a specified program (without compilation errors), as shown in the figure below:



4.5.6. Task Type

The Task Type is used for multi-task debugging, monitoring local variables, and displaying the running line number. After the program is successfully compiled or while it is running, you can view the values of local variables.

As shown in the figure, “Program1” is activated as Task 1. Select Task1 under Task Type to view the local variable values in Program1 and the program run cursor.

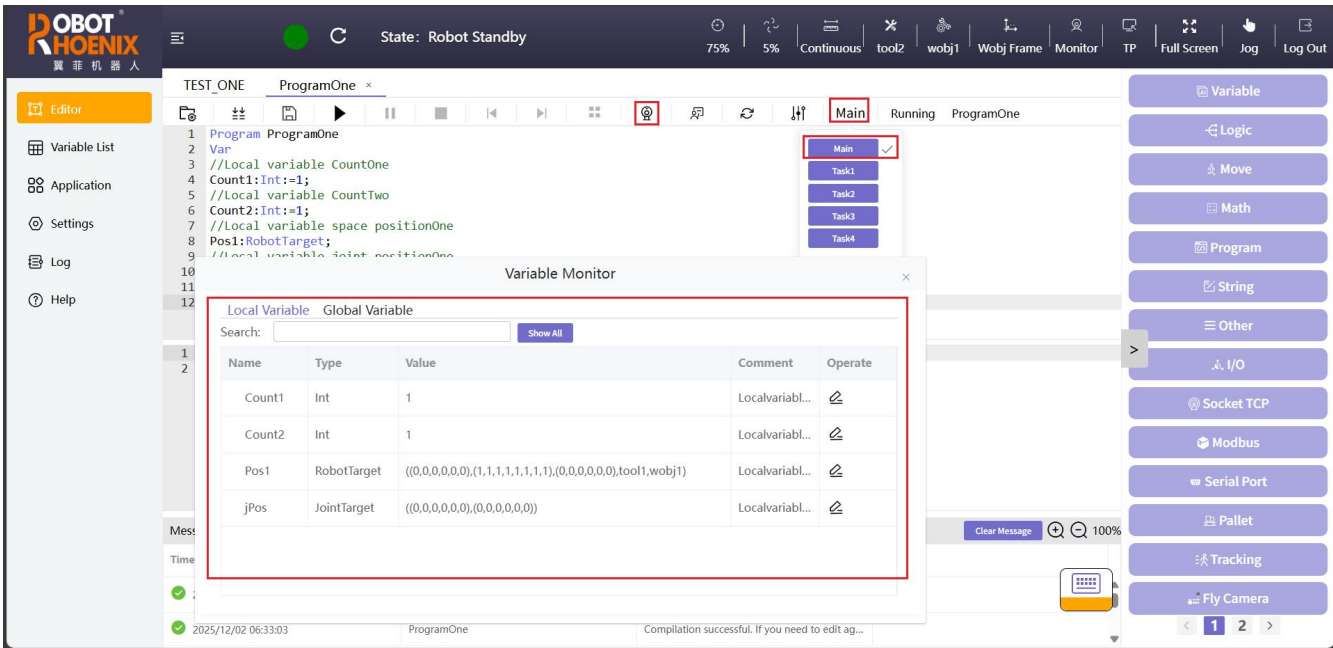


4.6. Variable Monitoring

4.6.1. Local Variable Monitoring

4.6.1.1. Overview

Local Variable Monitoring is used for monitoring variables during debugging and runtime. After the program is successfully compiled or while it is running, you can view or modify the values of local variables, as shown in the figure below:



Operation Method

Click the Variable Monitoring (🔍) button and select Local Variables to view or modify the values of all local variables defined in the program's Variable Declaration Area.

Note:

After the program is successfully compiled or while it is running, the values of local variables can be monitored.

For multi-task programs, you need to switch the Task Type (**Main**) to the corresponding task **Task1** (e.g., Task 1, Task 2, Task 3, or Task 4) to monitor local variables. For non-multi-task programs, switch the Task Type (**Main**) to Main Task to monitor local variables.

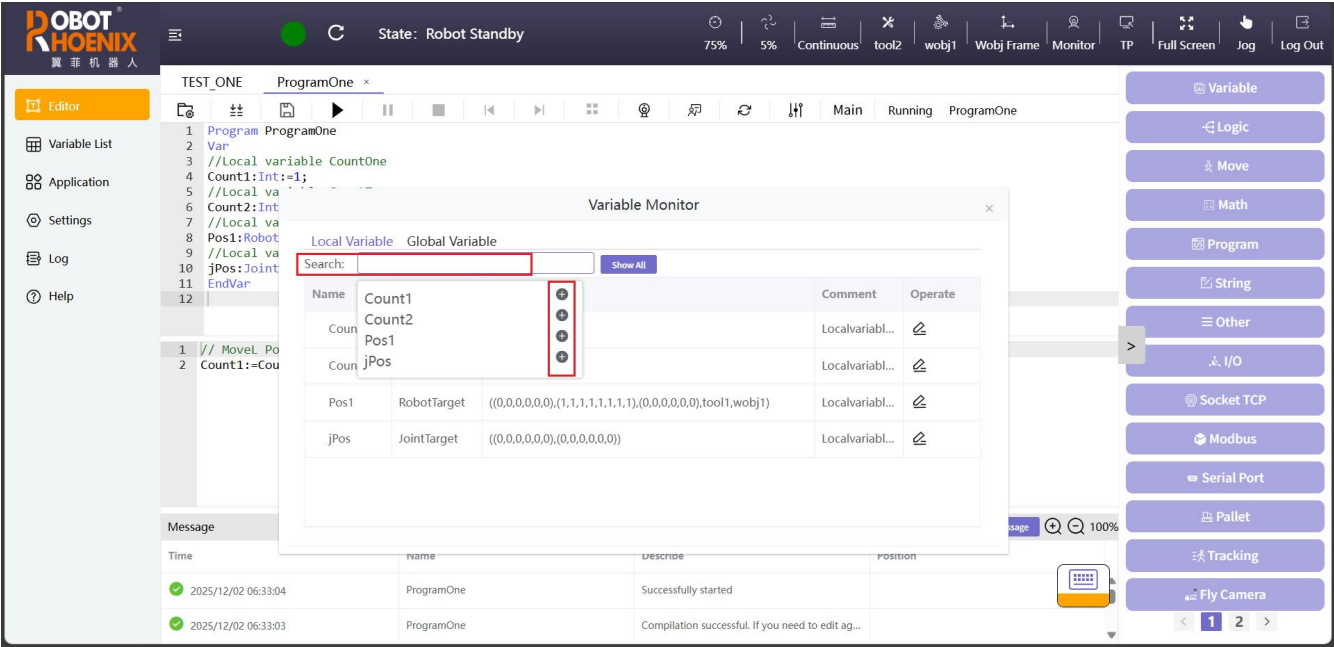
4.6.1.2. Add Monitored Variable

The Gorilla Studio robot control system supports adding monitored variables to enable monitoring of specific variables.

Operation Method

Click the search box in the Local Variable Monitoring interface. In the dropdown list, find the variable you want to monitor and click the corresponding Add (+) button to start monitoring.

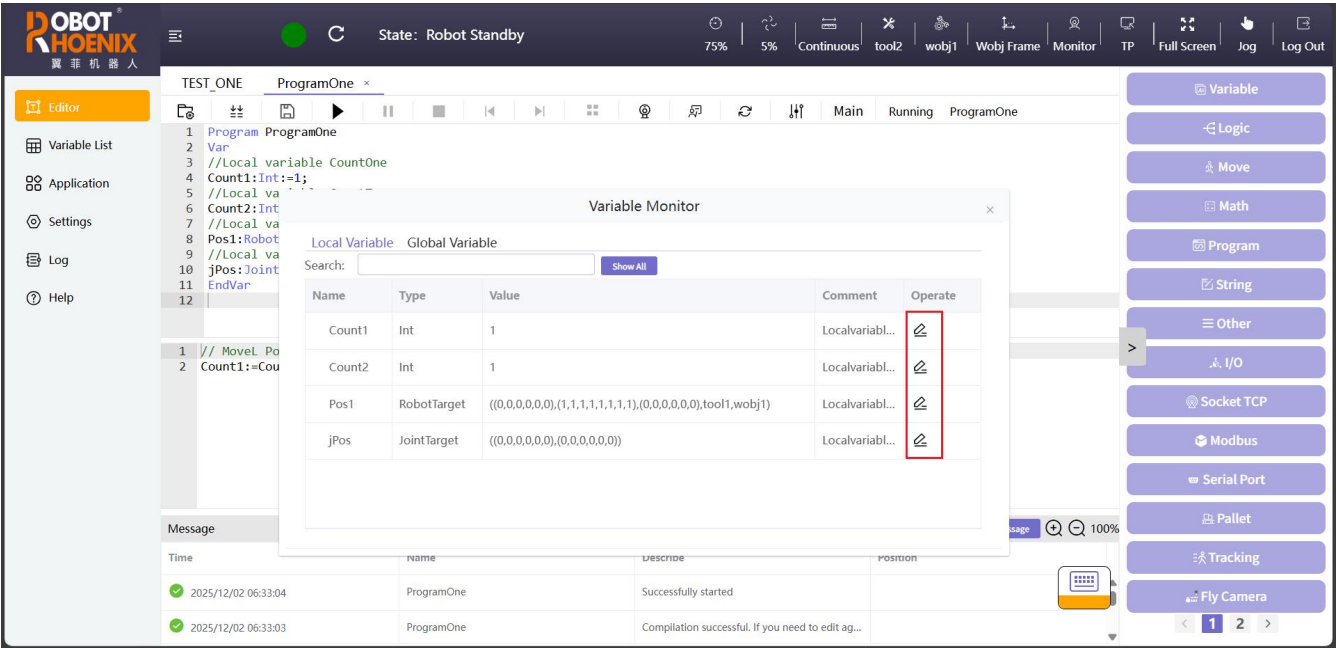
Multiple variables can be added. To stop monitoring a variable, click the corresponding Remove (⊖) button. Click the Show All button to display all local variables, as shown in the figure below:



4.6.1.3. Set Variable Value

Operation Method

Open the Local Variable Monitoring interface, click the operation (✎) button corresponding to the variable you want to set, enter the variable value, and click OK. As shown in the figure below:



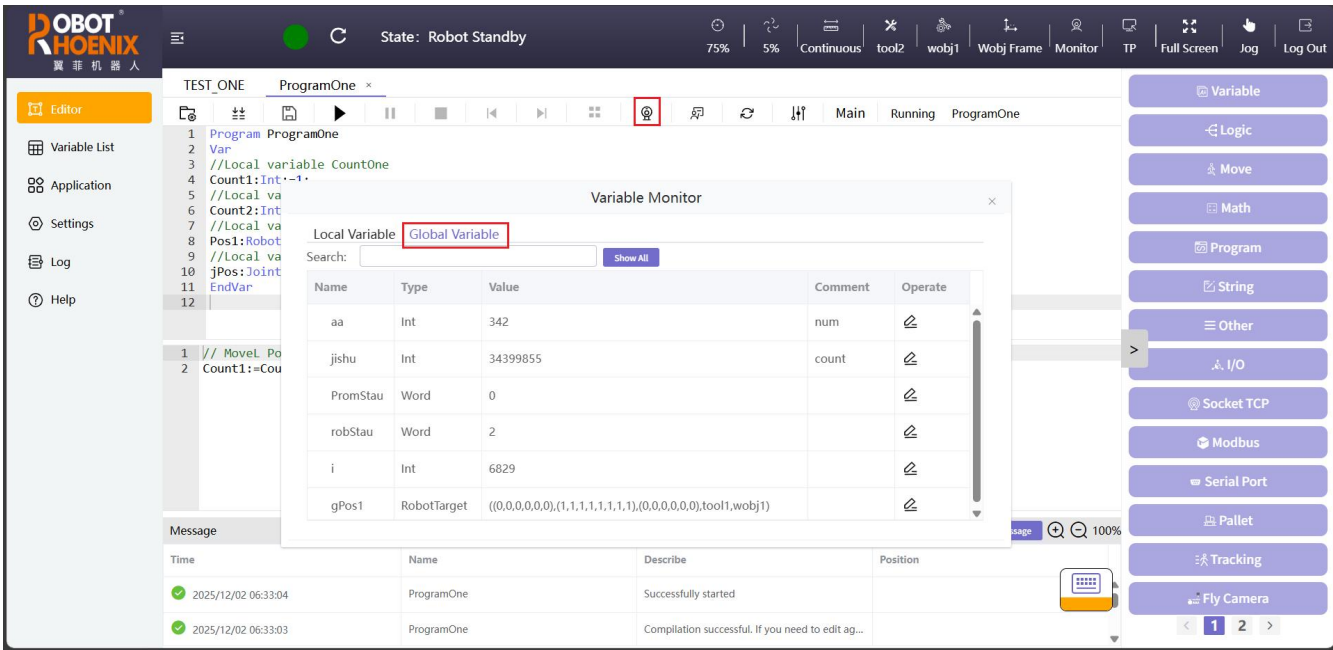
Note:

During program execution, it is permissible to modify RobotTarget and JointTarget type variable values. However, improper modifications may lead to collisions or other hazards, so please proceed with caution.

4.6.2. Global Variable Monitoring

4.6.2.1. Overview

Global Variable Monitoring provides real-time monitoring of the global variables defined in the Variable Table, as shown in the figure below:



Operation Method

Click the Variable Monitoring (🔍) button and select Global Variables to view or modify the values of all global variables defined in the Variable Table.

Note:

Unlike local variable monitoring, global variables can be monitored even when the program is not compiled or if there are compilation errors.

4.6.2.2. Add Monitored Variable

The Gorilla Studio robot control system supports adding monitored variables to enable monitoring of specific variables.

Operation Method

Click the search box in the Global Variable Monitoring interface. In the dropdown list, find the variable you want to monitor and click the corresponding Add (+) button to start monitoring. Multiple variables can be added. To stop monitoring a variable, click the corresponding Remove (−) button. Click the Show All button to display all global variables.

4.6.2.3. Set Variable Value

Operation Method

Open the Global Variable Monitoring interface, click the operation () button corresponding to the variable you want to set, enter the variable value, and click OK.

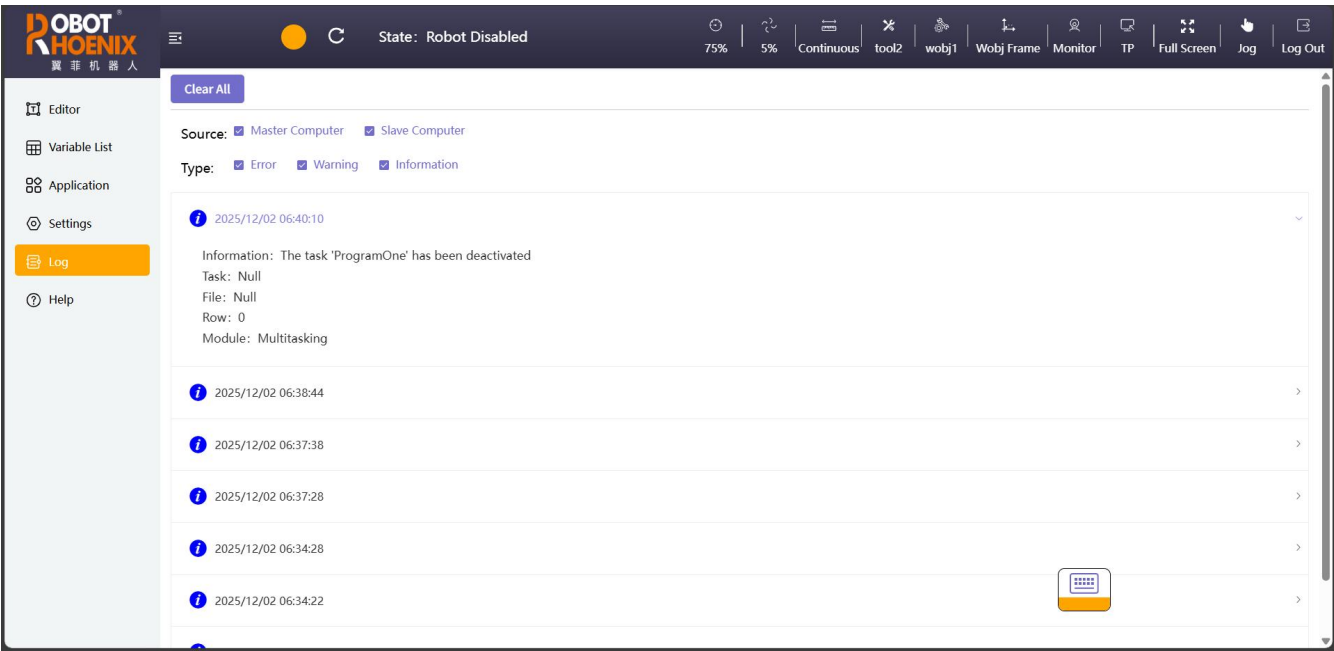
Note:

The values of global variables can also be set in Variable Table → Global Variables.

During program execution, it is allowed to modify RobotTarget and JointTarget type variable values. Improper modifications may cause collisions or other hazards, so please operate with caution.

5. Logs

The Gorilla Studio robot control system provides detailed operation logs, which can be used to trace the robot’s running status and identify the causes of faults. Click on a log date to expand the detailed content, which includes program information, task, program name, line number, and the associated module, as shown in the figure below:

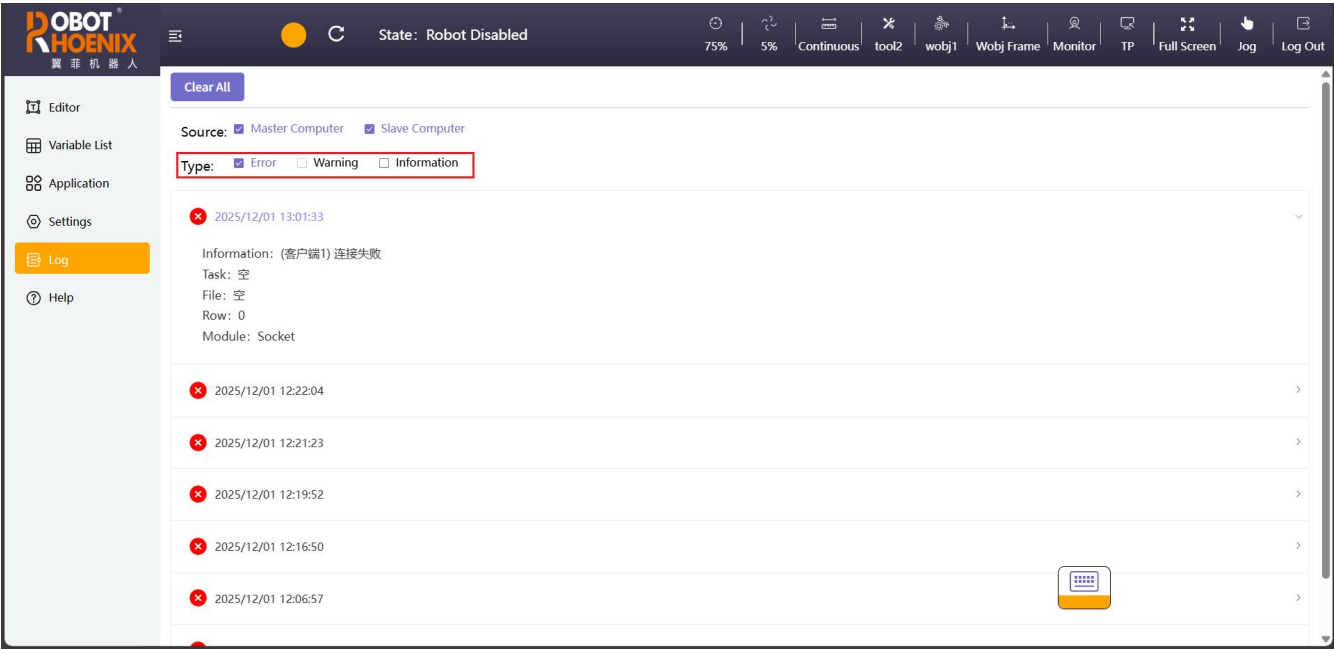


5.1. Filter Logs

Logs can be categorized by source into Host Logs and Controller Logs, and by type into Error, Warning, and Information.

Operation Method

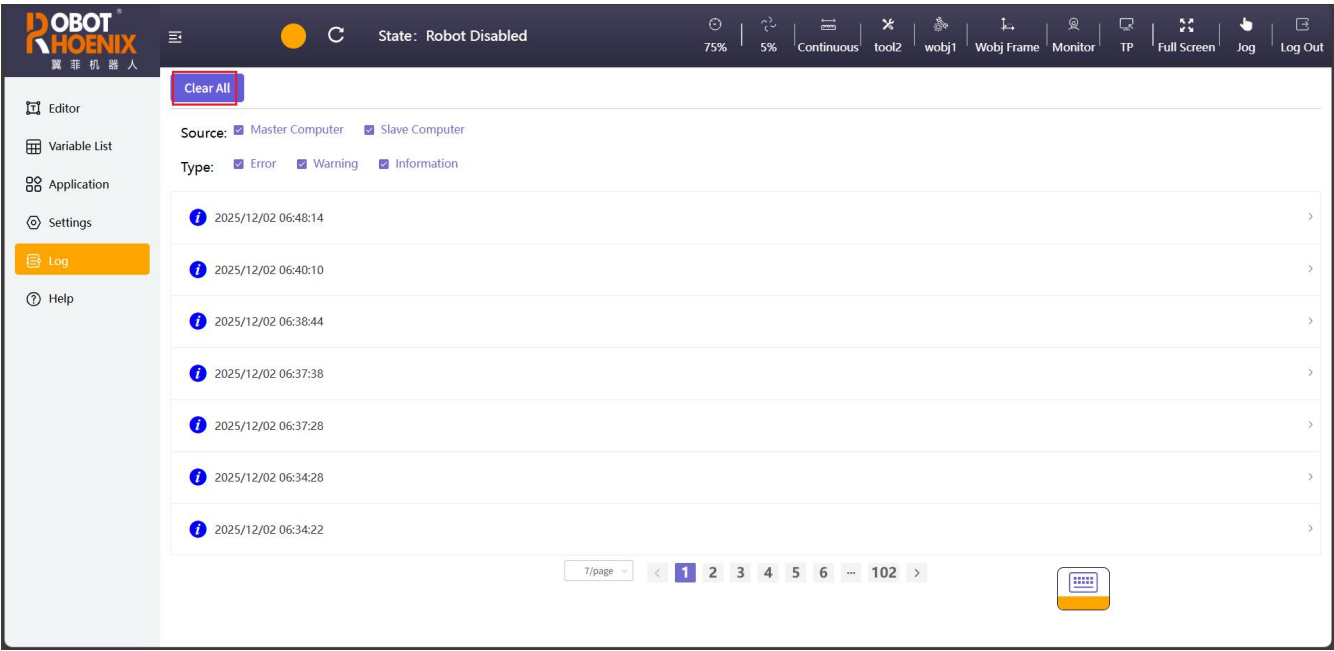
Open the log interface and select the log sources and types you want to view, as shown in the figure below:



5.2. Clear Logs

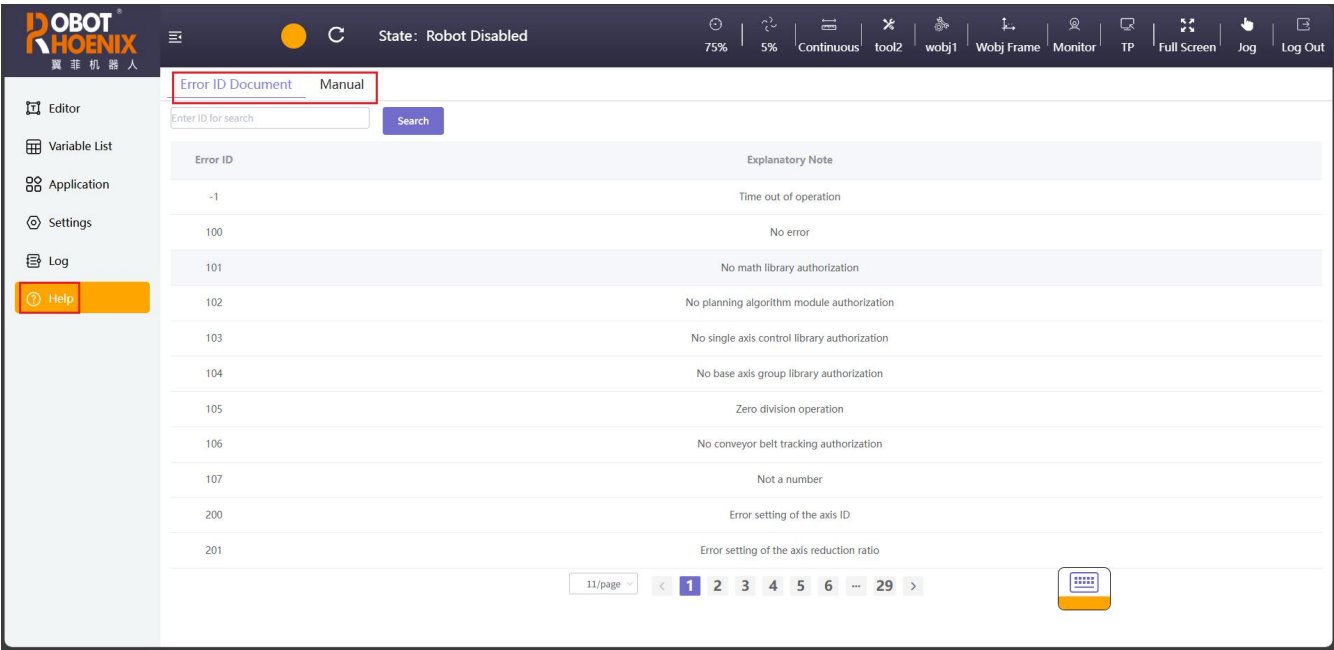
Operation Method

Open the log interface and click the Clear All button to delete all log information. This action cannot be undone, so please proceed with caution.



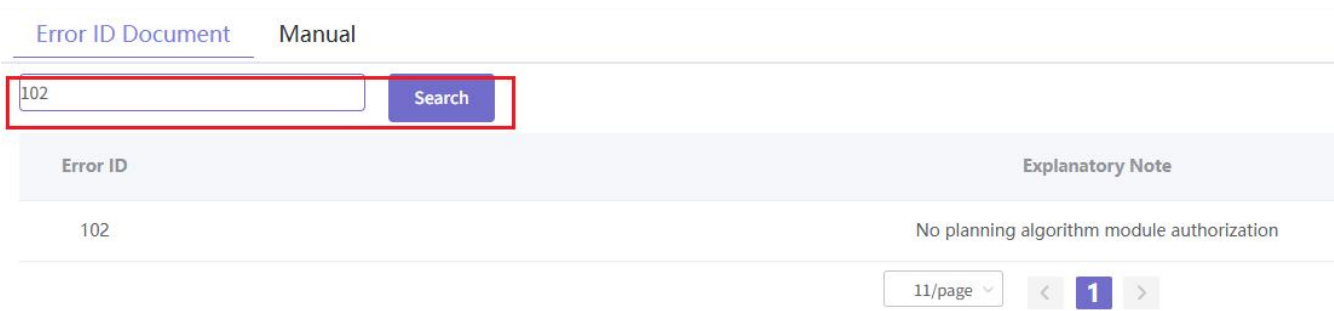
6. Help

The Help interface provides the error ID documentation and user manual. As shown in the figure below:



6.1. Error ID Documentation

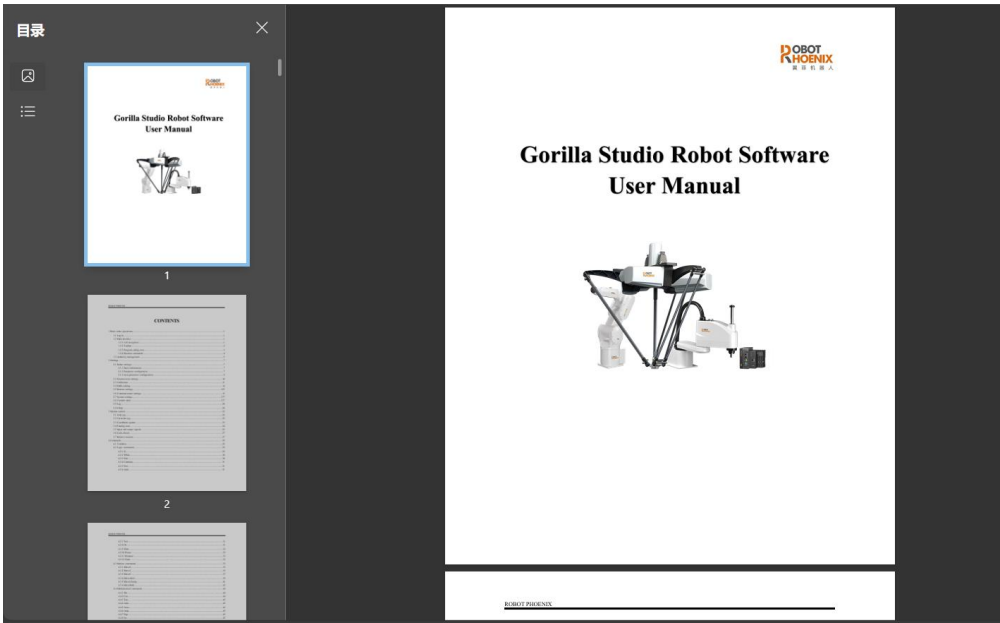
Gorilla Studio Robot Control System supports the error ID search function. Enter the error ID in the search box and click the "Search" button to find the corresponding information. As shown in the figure below:



6.2. Manual

Click "Manual" to open the *Gorilla Studio Robot Software User Manual*. As shown in the

figure below:



7. Command

7.1. Variables

7.1.1. Basic Types

7.1.1.1. Int

Description

Integer, with a range of allowed values from -2147483647 to 2147483647.

7.1.1.2. Double

Description

Double-precision floating-point type, with a value range from -99999999999.999 to 99999999999.999.

7.1.1.3. Float

Description

Single-precision floating-point type, with a value range from -99999999999.999 to 99999999999.999.

7.1.1.4. Bool

Description

Boolean type, mainly used for status or logical conditions, with values of True or False.

7.1.1.5. String

Description

String type, composed of multiple letters or numbers, and must be enclosed in single quotes (' ') when defined. The value can be up to 1000 characters long.

7.1.1.6. Word

Description

Unsigned short integer type, with a value range from 0 to 65535.

7.1.2. Complex Types

7.1.2.1. Vel

Description

Velocity, used to define the movement speed of the robot and external axes.

Internal format

(tcp, joint, rotation, jumpUp, jumpDown)

- tcp (%): Cartesian space velocity percentage (for MoveLJump motion, this parameter only applies to the horizontal segment). Data type: Double, allowed range: 1–100.
- joint (%): Joint space velocity percentage. Data type: Double, allowed range: 1–100.
- rotation (%): Cartesian rotation speed percentage. Data type: Double, allowed range: 1–100.
- jumpUp (%): Upward segment speed, i.e., the upward speed percentage in a MoveLJump instruction. Data type: Double, allowed range: 1–100.
- jumpDown (%): Downward segment speed, i.e., the downward speed percentage in a MoveLJump instruction. Data type: Double, allowed range: 1–100.

7.1.2.2. Acc

Description

Acceleration, used to define the movement acceleration of the robot and external axes.

Internal format

(tcp, joint, rotation, jumpUp, jumpDown)

- tcp (%): Cartesian space acceleration percentage (for MoveLJump motion, this parameter only applies to the horizontal segment). Data type: Double, allowed range: 1–100.
- joint (%): Joint space acceleration percentage. Data type: Double, allowed range: 1–100.
- rotation (%): Cartesian rotation acceleration percentage. Data type: Double, allowed range: 1–100.
- jumpUp (%): Upward segment acceleration, i.e., the upward acceleration percentage in a MoveLJump instruction. Data type: Double, allowed range: 1–100.
- jumpDown (%): Downward segment acceleration, i.e., the downward acceleration percentage in a MoveLJump instruction. Data type: Double, allowed range: 1–100.

7.1.2.3. Zone

Description

Corner zone, defines how the transition occurs between two motion trajectories.

Internal format

(distance, percent)

- distance (mm) : Cartesian space corner zone, data type is double, with a value range from 0 to 1999 mm.
- percent (%) : Joint space corner zone, data type is double, with a value range from 0 to 100%.

7.1.2.4. Pose

Description

Pose, stores the position and orientation in Cartesian space, with orientation represented using ZYX Euler angles.

Internal format

(X,Y,Z,Rx,Ry,Rz)

- X (mm) : The data type is double, with a value range from -9999 to 9999.

- Y (mm) : The data type is double, with a value range from -9999 to 9999.
- Z (mm) : The data type is double, with a value range from -9999 to 9999.
- Rx (°) : The data type is double, with a value range from -9999 to 9999.
- Ry (°) : The data type is double, with a value range from -9999 to 9999.
- Rz (°) : The data type is double, with a value range from -9999 to 9999.

7.1.2.5. AxisPos

Description

Axis position, stores the angles of each robot axis.

Internal format

(J1,J2,J3,J4,J5,J6)

- J1 (°) : 1 Axis position, The data type is double, with a value range from -180 to 180.
- J2 (°) : 2 Axis position, The data type is double, with a value range from -180 to 180.
- J3 (°) : 3 Axis position, The data type is double, with a value range from -9999 to 9999.
- J4 (°) : 4 Axis position, The data type is double, with a value range from -999999999 to 999999999.
- J5 (°) : 5 Axis position, The data type is double, with a value range from -360 to 360.
- J6 (°) : 6 Axis position, The data type is double, with a value range from -360 to 360.

7.1.2.6. RobotTarget

Description

Cartesian space target point, used for MoveJ, MoveL, MoveC, MoveLJump, and MovePath instructions.

Internal format

(pose, confData, eAxisPos, tool, wobj)

- pose (Target pose) :

X (mm) : The data type is double, with a value range from -9999 to 9999.

Y (mm) : The data type is double, with a value range from -9999 to 9999.

Z (mm) : The data type is double, with a value range from -9999 to 9999.

Rx (°) : The data type is double, with a value range from -9999 to 9999.

Ry (°) : The data type is double, with a value range from -9999 to 9999.

Rz (°) : The data type is double, with a value range from -9999 to 9999.

● confData (Configuration Data) :

1 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -4 to 3.

2 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -4 to 3.

3 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -300 to 300.

4 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -300 to 300.

5 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -4 to 3.

6 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -4 to 3.

7 Axis Rotation Quadrant: The data type is Int. The allowed value range is from -4 to 3.

Pose Type: The data type is Int, with an allowed value range from 0 to 8.

Description:

The axis rotation quadrant:

0 represents the axis angle range of [0, 90].

1 represents the axis angle range of (90, 180].

-1 represents the axis angle range of (0, -90]

● eAxisPos (External Axis position) :

E1(mm): External 1 Axis position, The data type is double, with a value range from -9999 to 9999.

E2(mm): External 2 Axis position, The data type is double, with a value range from -9999 to 9999.

E3(mm): External 3 Axis position, The data type is double, with a value range from -9999 to

9999.

E4(mm): External 4 Axis position, The data type is double, with a value range from -9999 to 9999.

E5(mm): External 5 Axis position, The data type is double, with a value range from -9999 to 9999.

E6(mm): External 6 Axis position, The data type is double, with a value range from -9999 to 9999.

- **Tool (Tool Coordinate System):** The data type is tool, with an allowed value range from tool1 to tool16.

- **Wobj (Workpiece Coordinate System):** The data type is wobj, with an allowed value range from wobj1 to wobj16.

7.1.2.7. JointTarget

Description

Joint space target point, used for the **MoveAbsJ** command.

Internal format

(jointPos, eAxisPos)

- jointPos (Axis position) :

J1 (°) : 1 Axis position, The data type is double, with a value range from -180 to180.

J2 (°) : 2 Axis position, The data type is double, with a value range from -180 to180.

J3 (°) : 3 Axis position, The data type is double, with a value range from -9999 to 9999.

J4 (°) : 4 Axis position, The data type is double, with a value range from -999999999 to 999999999.

J5 (°) : 5 Axis position, The data type is double, with a value range from -360 to 360.

J6 (°) : 6 Axis position, The data type is double, with a value range from -360 to 360.

- eAxisPos (External Axis position) :

E1(mm): External 1 Axis position, The data type is double, with a value range from -9999 to 9999.

E2(mm): External 2 Axis position, The data type is double, with a value range from -9999 to 9999.

E3(mm): External 3 Axis position, The data type is double, with a value range from -9999 to 9999.

E4(mm): External 4 Axis position, The data type is double, with a value range from -9999 to 9999.

E5(mm): External 5 Axis position, The data type is double, with a value range from -9999 to 9999.

E6(mm): External 6 Axis position, The data type is double, with a value range from -9999 to 9999.

7.1.2.8. JumpData

Description

Gate-shaped motion Parameters.

Internal format

(upDis, upPer, downDis, downPer, rotUpDis, rotDownDis)

- upDis (Rising Distance) (mm): The rising distance for gate-shaped motion. A positive value (+) represents movement upward along the Z-axis of the specified workpiece coordinate system, while a negative value (-) represents movement downward along the Z-axis of the specified workpiece coordinate system. The data type is double, with a value range from -999 to 999.
- upPer (Rising Corner Zone Percentage) (%): The percentage of the rising corner zone in gate-shaped motion. The data type is double, with a value range from 0 to 100.
- downDis (Descending Distance) (mm): The descending distance for gate-shaped motion. A positive value (+) represents movement upward along the Z-axis of the specified workpiece coordinate system, while a negative value (-) represents movement downward along the Z-axis. The data type is double, with a value range from -999 to 999.
- downPer (Descending Corner Zone Percentage) (%): The percentage of the descending corner zone in gate-shaped motion. The data type is double, with a value range from 0 to 100.
- rotUpDis (Posture Start Height) (mm): The posture start height for gate-shaped motion. A

positive value (+) represents movement upward along the Z-axis of the specified workpiece coordinate system, while a negative value (-) represents movement downward along the Z-axis. The data type is double, with a value range from -999 to 999.

- **rotDownDis (Posture Stop Height) (mm):** The posture stop height for gate-shaped motion. A positive value (+) represents movement upward along the Z-axis of the specified workpiece coordinate system, while a negative value (-) represents movement downward along the Z-axis. The data type is double, with a value range from -999 to 999.

7.2. Logical Command

7.2.1. If

Description

Conditional statement, if the condition is met, the corresponding statement is executed.

Format

```
If Condition1 Then  
    Stmt1;  
  
ElseIf Condition2 Then  
    Stmt2;  
  
Else  
    Stmt3;  
  
EndIf
```

Parameters

Condition: Represents the condition.

Stmt: The statement executed if the condition is met

Description:

- The condition Condition can be a combination of multiple conditions.
- The statement Stmt can consist of multiple lines of instructions.

- If is used to begin the block, Else is optional, and EndIf is required to mark the end of the block.

Example

```
If (bool1 = True) And (int1 > 1) Then
```

```
    MoveL pose1,vel1,zone1;
```

```
ElseIf int1 < 1 Then
```

```
    MoveJ pose1,vel1,zone1;
```

```
Else
```

```
    MovePath pose1,vel1,zone1;
```

```
EndIf
```

7.2.2. For**Description**

First, execute an initialization assignment statement. Then, execute the contents of the loop body. After completing one iteration, increment the variable in the initialization assignment statement by the step size. Repeat the steps until the condition is no longer met, at which point the loop exits.

Format

```
For i := InitValue To TargetValue By Step Do
```

```
    Stmt;
```

```
EndFor
```

Parameters

InitValue: The initial value.

TargetValue: The target value.

Step: The step size, which defines the increment of the corresponding variable after each iteration of the loop.

Stmt: The statement executed when the condition is met.

Description:

- The statement Stmt can consist of multiple lines of instructions.

Example

```
For m := 1 To 32 By 1 Do
```

```
    SetDO(m,False);
```

```
EndFor
```

7.2.3. While

Description

The condition is evaluated, and if the condition is met, the statements within the loop body are executed. Once the condition is no longer satisfied, the loop exits.

Format

```
While Condition Do
```

```
    Stmt;
```

```
EndWhile
```

Parameters

Condition: Represents the condition.

Stmt: The statement executed if the condition is met.

Description:

- The Condition can be a combination of multiple conditions.
- The Stmt can consist of multiple lines of instructions.

Example

```
While int1 < 3 Do
```

```
    MoveL pose1,vell,zone1;
```

```
    MoveL pose2,vell,zone1;
```

```
    int1 := int1+1;
```

EndWhile

7.2.4. Goto

Description

Label is used to set a program label, commonly used in conjunction with the **Goto** command to perform a jump action.

Format

Label:: //The label name must be unique and cannot be duplicated.

Goto(Label);

Example

```
MoveL pose0,vel1,zone1;
```

```
Label1::                    //Set Label1
```

```
MoveL pose1,vel1,zone1;
```

```
MoveL pose2,vel1,zone1;
```

```
Goto(Label1);              //Go to Label1
```

//Execution effect of this section: First, the program moves to pose0, then it performs a back-and-forth motion between pose1 and pose2.

Note

- The label name can be customized, such as Label1 in the example.
- A program can have multiple labels, but label names must be unique.
- Label and Goto must be within the same scope.

7.2.5. Exit

Description

The **Exit** statement is used to exit the current loop. It is typically used in a **While** loop. When the loop reaches **Exit**, it will immediately exit the **While** loop, regardless of the loop's

condition. In the case of nested loops, it only affects the current level of the loop.

Format

Exit;

Example

int1 := 3;

While bool1 = True Do

 MoveL pose1,vel1,zone1;

 MoveL pose2,vel1,zone1;

 int1 := int1-1;

 If int1 <= 1 Then

 Exit;

 EndIf

EndWhile

//Execution effect of this section: It will execute **MoveL pose1** and **MoveL pose2** two times.

7.2.6. Continue

Description

The Continue statement skips the remaining statements in the current iteration of the While loop and moves to the next iteration. In the case of nested loops, it only affects the current level of the loop.

Format

Continue;

Example

int1 := 3;

While bool1 = True Do

 MoveL pose1,vel1,zone1;


```
    MoveL pose2,vel1,zone1;

    int1 := int1-1;

    If int1 <= 1 Then

        Continue;

    EndIf

    MoveL pose3,vel1,zone1;

EndWhile

//Execution effect of this section: MoveL pose3 will be executed only once.
```

7.2.7. Not

Description

Logical NOT, negation.

7.2.8. And

Description

Logical AND.

7.2.9. Xor

Description

Logical XOR (Exclusive OR).

7.2.10. Or

Description

Logical OR.

7.2.11. Wait

Description

Signal Wait: Waits until the input or output signal meets the specified condition.

The Wait instruction is used to wait for the occurrence of a specific event or time. Otherwise, the task remains in a waiting state.

Format

- Wait (time): Waits for the specified time before executing the next instruction. The unit is seconds (s).
- Wait (expression): Waits for the expression to evaluate as True before executing the next instruction.

Example

```
Wait(2);
```

```
//Wait for 2 seconds before executing the subsequent program.
```

```
Wait(DI1 = True And DI2 = True);
```

```
//Wait until both DI1 and DI2 are True before executing the subsequent program.
```

7.2.12. Pause

Description

Format

```
Pause;
```

Note: The Pause instruction in multi-tasking will pause the main task.

7.2.13. Resume

Description

Resume (restart) the task.

Format

Resume;

7.2.14. Case

Description

Conditional selection statement: Based on the value of the expression, the program selects one branch to execute from multiple options.

Format

Case Var Of

Value1:

 Stmt1;

Value2:

 Stmt2;

Value3,Value4,Value5:

 Stmt3;

Else

 Stmt4;

EndCase

Parameters

Var : Condition Variable

Value1 : Label

Stmt: Statements that execute when the condition is satisfied

Description:

- If several values of the condition variable require the same instruction to be executed, these values can be written together, separated by commas.

The statement (Stmt) can consist of multiple lines of instructions

Example

Case m Of

1:

SetDO(1,False);

2:

SetDO(2,False);

Else

SetDO(3,False);

EndCase

7.3. Motion Command

7.3.1. MoveL

Description

Move in a straight line to the specified target position.

Format

MoveL [Offs/RelTool]RobotTarget,Vel,Zone,[Wobj],[Tool],[Acc],[Until DI = Value],[Out(DO,Output Value,Type,Type Value)],[Out(DO,Output Value,Type,Type Value)];

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **RobotTarget:** The target position described in Cartesian space. **Data Type:** RobotTarget

Common form: RobotTarget

Offset form: Offs(RobotTarget,X,Y,Z,Rx,Ry,Rz)\ RelTool(RobotTarget,X,Y,Z,Rx,Ry,Rz)

(1) **Offs:** Offset based on the workpiece coordinate system. X, Y, Z represent the offset values along the workpiece coordinate system, and Rx, Ry, Rz represent the offset values around the workpiece coordinate system.

(2) **RelTool:** Offset based on the tool coordinate system. X, Y, Z represent the offset

values along the tool coordinate system, and Rx, Ry, Rz represent the offset values around the tool coordinate system.

- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **Wobj:** (Optional parameter) The selected workpiece coordinate system, which can be one of **wobj1** to **wobj16**.

When this parameter is not set, the workpiece coordinate system in the target pose RobotTarget is used.

For example, the MoveL command in the image below uses the wobj4 workpiece coordinate system.

```
18 lp1111 : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveL lp1111,vel1,zone1;
```

- **Tool:** (Optional parameter) The selected tool coordinate system, which can be one of **tool1** to **tool16**.

When this parameter is not set, the tool coordinate system in the target pose RobotTarget is used.

For example, the MoveL command in the image below uses the tool2 tool coordinate system.

```
18 lp1111 : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveL lp1111,vel1,zone1;
```

- **Acc:** (Optional parameter) Specifies acceleration.

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

- **Until DI = Value:** (Optional parameter) Detects signal during execution; stops immediately when the condition is met.

During motion, the input signal is monitored. If the condition is met, the current motion is immediately aborted, and the program continues with the following lines. If the condition is not met, the motion continues until the endpoint.

Sub-parameters:

DI: Input signal, supporting only DI1-DI16.

Value: The input signal value, either **True** or **False**.

Example:

//Monitor the DI during motion. If it is True, the motion is immediately aborted; otherwise, the motion continues until the endpoint.

MoveL RobotTarget,vel1,zone1,Until DI1=True;

Note:

When using the Until parameter, the cornering zone setting will not take effect.

● **Out(DO,Output Value,Type,Type Value):** (Optional parameter) Output signals in parallel during execution.

A maximum of 2 signals can be output in parallel in a single motion instruction.

Sub-parameters:

DO: Output signals, supporting only DO1-DO16.

Output Value: Output signal value, either True or False.

Type: The type of signal output during motion. Different types of motion can use different Types.

Type Value: Type value.

Type	Meaning	Applicable Motion Types
Time(s)	Output signal based on time during motion: Type Value represents time, unit: seconds, range: -99 to 99. Type Value ≥ 0 means the signal will be output after the specified time from the start of motion. Type Value < 0 means the signal will be output a specified time before reaching the target point. When the Type Value exceeds the motion time range, the signal will be triggered only at the start and end points of that segment, and the specified time will be invalid.	MoveL、 MoveJ、 MoveC、 MoveAbsJ、 MoveLJump

Percentage(%)	Output signal based on path percentage during motion: Type Value represents the path percentage, indicating that the signal will be output when the motion reaches Type Value % of the total path. The range is from 0 to 100.	MoveL、 MoveJ、 MoveC、 MoveAbsJ、 MoveLJump
Distance(mm)	Output signal based on distance during motion: Type Value represents distance, unit: mm, range: -1999 to 1999. Type Value ≥ 0 means the signal will be output after moving the specified distance from the starting point. Type Value < 0 means the signal will be output when the motion reaches the specified distance from the endpoint.	MoveL、 MoveC、 MoveLJump

Note:

The motion I/O set within the cornering zone cannot guarantee accuracy.

When the specified Type Value exceeds the motion range, the signal will be triggered only at the start and end points of that segment.

Example:

//After 3 seconds of motion, DO1 outputs True. When there are 2 seconds remaining before the motion ends, DO2 outputs True.

```
MoveL RobotTarget,vel1,zone1,Out(DO1,True,T,3),Out(DO2,True,T,-2);
```

//When the motion reaches 10% of the path, DO1 outputs True. When the motion reaches 100% of the path, DO2 outputs True.

```
MoveL RobotTarget,vel1,zone1,Out(DO1,True,P,10),Out(DO2,True,P,100);
```

//After moving 50mm, DO1 outputs True. When there are 10mm remaining to the endpoint, DO2 outputs True.

```
MoveL RobotTarget,vel1,zone1,Out(DO1,True,D,50),Out(DO2,True,D,-10);
```

7.3.2. MoveJ

Description

Move the robot quickly from one point to another, with all axes reaching the target position simultaneously. The robot's end effector moves along an irregular curve. Please be cautious of any potential collision risks. This is used in cases where there are no specific requirements for the robot's end effector motion trajectory.

Format

MoveJ [Offs/RelTool]RobotTarget, Vel, Zone, [Wobj], [Tool], [Acc], [Until DI = Value], [Out(DO, Output Value, Type, Type Value)], [Out(DO, Output Value, Type, Type Value)];

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **RobotTarget:** The target position described in Cartesian space. **Data Type:** RobotTarget
Common form: RobotTarget
Offset form: Offs(RobotTarget,X,Y,Z,Rx,Ry,Rz)\ RelTool(RobotTarget,X,Y,Z,Rx,Ry,Rz)
(1) **Offs:** Offset based on the workpiece coordinate system. X, Y, Z are the offset values along the workpiece coordinate system, and Rx, Ry, Rz are the offset values around the workpiece coordinate system.
(2) **RelTool:** Offset based on the tool coordinate system. X, Y, Z are the offset values along the tool coordinate system, and Rx, Ry, Rz are the offset values around the tool coordinate system.
- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **Wobj:** (Optional parameter) The selected workpiece coordinate system, which can be one of wobj1 to wobj16.

When this parameter is not set, the workpiece coordinate system in the target pose RobotTarget is used.

For example, the MoveJ command in the image below uses the wobj4 workpiece coordinate system.


```
18 lp1111 : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveJ lp1111,vel1,zone1;
```

- **Tool: (Optional parameter)** The selected tool coordinate system, which can be one of tool1 to tool16.

When this parameter is not set, the tool coordinate system in the target pose RobotTarget is used.

For example, the MoveJ command in the image below uses the tool2 tool coordinate system.

```
18 lp1111 : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveJ lp1111,vel1,zone1;
```

- **Acc: (Optional parameter)** Specify acceleration.

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

- **Until DI = Value: (Optional parameter)** Detect signals during execution; stop immediately when the condition is met.

Monitor the input signal during motion. If the condition is met, the current motion is immediately aborted, and the program continues with the subsequent lines. If the condition is not met, the motion will continue until the endpoint.

Sub-parameters:

DI: Input signal, supporting only DI1 to DI16.

Value: Input signal value, either True or False.

Example:

//Monitor DI during motion. If it is True, the motion is immediately aborted; otherwise, the motion continues until the endpoint.

MoveJ RobotTarget,vel1,zone1,Until DI1=True;

Note:

When using the Until parameter, the cornering zone setting will not take effect.

- **Out(DO,Output Value,Type,Type Value): (Optional parameter)** Output signals in parallel during execution.

A maximum of 2 signals can be output in parallel in a single motion instruction.

Sub-parameters :

DO: Output signals, supporting only DO1-DO16.

Output Value: Output signal value, either True or False.

Type: The type of signal output during motion. Different types of motion can use different Types.

Type Value: Type value.

Type	Meaning	Applicable Motion Types
Time(s)	Output signal based on time during motion: Type Value represents time, unit: seconds, range: -99 to 99. Type Value ≥ 0 means the signal will be output after the specified time from the start of motion. Type Value < 0 means the signal will be output a specified time before reaching the target point. When the Type Value exceeds the motion time range, the signal will be triggered only at the start and end points of that segment, and the specified time will be invalid.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Percentage(%)	Output signal based on path percentage during motion: Type Value represents the path percentage, indicating that the signal will be output when the motion reaches Type Value % of the total path. The range is from 0 to 100.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Distance(mm)	Output signal based on distance during motion: Type Value represents distance, unit: mm, range: -1999 to 1999. Type Value ≥ 0 means the signal will be output after moving the specified distance from the starting point. Type Value < 0 means the signal will be output when the motion reaches the specified distance from the endpoint.	MoveL、MoveC、MoveLJump

Note:

The accuracy of motion I/O set within the cornering zone cannot be guaranteed.

When the set Type Value exceeds the motion range, it will trigger only at the start and end points of the travel range.

Example:

//After 3 seconds of motion, DO1 outputs True. When there are 2 seconds remaining before the motion ends, DO2 outputs True.

```
MoveJ RobotTarget,vel1,zone1,Out(DO1,True,T,3),Out(DO2,True,T,-2);
```

//When the motion reaches 10% of the path, DO1 outputs True. When the motion reaches 100% of the path, DO2 outputs True.

```
MoveJ RobotTarget,vel1,zone1,Out(DO1,True,P,10),Out(DO2,True,P,100);
```

7.3.3. MoveC

Description

Move along an arc through the intermediate auxiliary point to reach the given target position.

Format

```
MoveC AuxilliaryPoint ,TargetPoint,Vel,Zone,[Wobj],[Tool],[Acc],[Until DI =  
Value],[Out(DO,Output Value,Type,Type Value)],[Out(DO,Output Value,Type,Type Value)];
```

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **TargetPoint:** Target position described in Cartesian space. **Data Type:** RobotTarget
- **AuxilliaryPoint :** Intermediate point position described in Cartesian space, used to determine the arc size and direction of motion. **Data Type:** RobotTarget
- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **Wobj:** (Optional parameter) The selected workpiece coordinate system, which can be one of wobj1 to wobj16.

When this parameter is not set, the workpiece coordinate system in the target pose TargetPoint will be used.

For example, in the following diagram, the MoveC instruction uses the wobj4 workpiece coordinate system.

```

18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19 auxiliarypoint : RobotTarget:=((-652.9393,120,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool3,wobj5);
20
1 MoveC auxiliarypoint,targetpoint,vel1,zone1;
2

```

- **Tool: (Optional parameter)** The selected tool coordinate system, which can be one of tool1 to tool16.

When this parameter is not set, the tool coordinate system in the target pose TargetPoint will be used.

For example, in the following diagram, the MoveC instruction uses the tool2 tool coordinate system.

```

18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19 auxiliarypoint : RobotTarget:=((-652.9393,120,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool3,wobj5);
20
1 MoveC auxiliarypoint,targetpoint,vel1,zone1;
2

```

- **Acc: (Optional parameter)** Specify acceleration.

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

- **Until DI = Value: (Optional parameter)** Detect signals during execution; stop immediately when the condition is met.

Monitor the input signal during motion. If the condition is met, the current motion is immediately aborted, and the program continues with the subsequent lines. If the condition is not met, the motion will continue until the endpoint.

Sub-parameters:

DI: Input signal, supporting only DI1 to DI16.

Value: Input signal value, either True or False.

Example:

//Monitor DI during motion. If it is True, the motion is immediately aborted; otherwise, the motion continues until the endpoint.

MoveC AuxilliaryPoint , TargetPoint,vel1,zone1,Until DI1=True;

Note:

When using the Until parameter, the cornering zone setting will not take effect.

- **Out(DO,Output Value,Type,Type Value): (Optional parameter) Output signals in parallel during execution.**

A maximum of 2 signals can be output in parallel in a single motion instruction.

Sub-parameters :

DO: Output signals, supporting only DO1-DO16.

Output Value: Output signal value, either True or False.

Type: The type of signal output during motion. Different types of motion can use different Types.

Type Value: Type value.

Type	Meaning	Applicable Motion Types
Time(s)	Output signal based on time during motion: Type Value represents time, unit: seconds, range: -99 to 99. Type Value ≥ 0 means the signal will be output after the specified time from the start of motion. Type Value < 0 means the signal will be output a specified time before reaching the target point. When the Type Value exceeds the motion time range, the signal will be triggered only at the start and end points of that segment, and the specified time will be invalid.	MoveL、 MoveJ、 MoveC、 MoveAbsJ、 MoveLJump
Percentage(%)	Output signal based on path percentage during motion: Type Value represents the path percentage, indicating that the signal will be output when the motion reaches Type Value % of the total path. The range is from 0 to 100.	MoveL、 MoveJ、 MoveC、 MoveAbsJ、 MoveLJump
Distance(mm)	Output signal based on distance during motion: Type Value represents distance, unit: mm, range: -1999 to 1999. Type Value ≥ 0 means the signal	MoveL、 MoveC、 MoveLJump

	will be output after moving the specified distance from the starting point. Type Value < 0 means the signal will be output when the motion reaches the specified distance from the endpoint.	
--	--	--

Note:

The accuracy of motion I/O set within the cornering zone cannot be guaranteed.

When the set Type Value exceeds the motion range, it will trigger only at the start and end points of the stroke.

Example:

//After 3 seconds of motion, DO1 outputs True. When there are 2 seconds remaining before the motion ends, DO2 outputs True.

```
MoveC AuxilliaryPoint ,TargetPoint,vel1,zone1,Out(DO1,True,T,3),Out(DO2,True,T,-2);
```

//When the motion reaches 10% of the path, DO1 outputs True. When the motion reaches 100% of the path, DO2 outputs True.

```
MoveC AuxilliaryPoint ,TargetPoint,vel1,zone1,Out(DO1,True,P,10),Out(DO2,True,P,100);
```

//After moving 50mm, DO1 outputs True. When there are 10mm remaining to the endpoint, DO2 outputs True.

```
MoveC AuxilliaryPoint ,TargetPoint,vel1,zone1,Out(DO1,True,D,50),Out(DO2,True,D,-10);
```

7.3.4. MoveAbsJ

Description

Move both the Robot and External to a position defined by axis angles. This is used for quick positioning or moving the Robot to a precise axis angle. All axes move synchronously, and the Robot's end effector follows an irregular curve. Please note if there is any risk of collision.

Format

```
MoveAbsJ JointTarget,Vel,Zone,[Acc],[Until DI = Value],[Out(DO,Output Value,Type,Type Value)],[Out(DO,Output Value,Type,Type Value)];
```

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **JointTarget:** Target joint angles. **Data Type:** JointTarget
- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **Acc:** (Optional parameter) Specify acceleration.

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

- **Until DI = Value:** (Optional parameter) Detect signals during execution; stop immediately when the condition is met.

Monitor the input signal during motion. If the condition is met, the current motion is immediately aborted, and the program continues with the subsequent lines. If the condition is not met, the motion will continue until the endpoint.

Sub-parameters:

DI: Input signal, supporting only DI1 to DI16.

Value: Input signal value, either True or False.

Example:

//Monitor DI during motion. If it is True, the motion is immediately aborted; otherwise, the motion continues until the endpoint.

MoveAbsJ JointTarget,vel1,zone1,Until DI1=True;

Note:

When using the Until parameter, the cornering zone setting will not take effect.

- **Out(DO,Output Value,Type,Type Value):** (Optional parameter) Output signals in parallel during execution.

A maximum of 2 signals can be output in parallel in a single motion instruction.

Sub-parameters:

Sub-parameters:

DO: Output signals, supporting only DO1-DO16.

Output Value: Output signal value, either True or False.

Type: The type of signal output during motion. Different types of motion can use different Types.

Type Value: Type value.

Type	Meaning	Applicable Motion Types
Time(s)	Output signal based on time during motion: Type Value represents time, unit: seconds, range: -99 to 99. Type Value ≥ 0 means the signal will be output after the specified time from the start of motion. Type Value < 0 means the signal will be output a specified time before reaching the target point. When the Type Value exceeds the motion time range, the signal will be triggered only at the start and end points of that segment, and the specified time will be invalid.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Percentage(%)	Output signal based on path percentage during motion: Type Value represents the path percentage, indicating that the signal will be output when the motion reaches Type Value % of the total path. The range is from 0 to 100.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Distance(mm)	Output signal based on distance during motion: Type Value represents distance, unit: mm, range: -1999 to 1999. Type Value ≥ 0 means the signal will be output after moving the specified distance from the starting point. Type Value < 0 means the signal will be output when the motion reaches the specified distance from the endpoint.	MoveL、MoveC、MoveLJump

Note:

The accuracy of motion I/O set within the cornering zone cannot be guaranteed.

When the set Type Value exceeds the motion range, it will trigger only at the start and end points of the stroke.

Example:

//After 3 seconds of motion, DO1 outputs True. When there are 2 seconds remaining before the

motion ends, DO2 outputs True.

```
MoveAbsJ JointTarget,vel1,zone1,Out(DO1,True,T,3),Out(DO2,True,T,-2);
```

//When the motion reaches 10% of the path, DO1 outputs True. When the motion reaches 100% of the path, DO2 outputs True.

```
MoveAbsJ JointTarget,vel1,zone1,Out(DO1,True,P,10),Out(DO2,True,P,100);
```

7.3.5. MoveLJump

Description

Gate-shaped motion. The motion process is divided into three segments: the initial ascending segment (MoveL), the middle segment (MoveL), and the final descending segment (MoveL).

Format

```
MoveLJump [Offs/RelTool]RobotTarget,Vel,Zone,JumpData,[Wobj],[Tool],[Acc],[Until DI = Value],[Out(DO,Output Value,Type,Type Value)],[Out(DO,Output Value,Type,Type Value)];
```

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **RobotTarget:** Target position described in Cartesian space. **Data Type:** RobotTarget
Common form: RobotTarget
Offset form: Offs(RobotTarget,X,Y,Z,Rx,Ry,Rz)、RelTool(RobotTarget,X,Y,Z,Rx,Ry,Rz)
(1) Offs: Offset based on the workpiece coordinate system. X, Y, Z are the offset values along the workpiece coordinate system, and Rx, Ry, Rz are the offset values around the workpiece coordinate system.
(2) RelTool: Offset based on the tool coordinate system. X, Y, Z are the offset values along the tool coordinate system, and Rx, Ry, Rz are the offset values around the tool coordinate system.
- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **JumpData:** Gate-shaped motion parameters.
- **Wobj:** (Optional parameter) The selected workpiece coordinate system, which can

be one of wobj1 to wobj16.

When this parameter is not set, the workpiece coordinate system in the target pose RobotTarget is used.

For example, in the following diagram, the MoveLJump instruction uses the wobj4 workpiece coordinate system.

```
18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveLJump targetpoint,vel1,zone1,lJumpData;
```

- **Tool: (Optional parameter) The selected tool coordinate system, which can be one of tool1 to tool16.**

When this parameter is not set, the tool coordinate system in the target pose RobotTarget is used.

For example, in the following diagram, the MoveLJump instruction uses the tool2 tool coordinate system.

```
18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MoveLJump targetpoint,vel1,zone1,lJumpData;
```

- **Acc: (Optional parameter) Specify acceleration.**

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

- **Until DI = Value: (Optional parameter) Detect signals during execution; stop immediately when the condition is met.**

Monitor the input signal during motion. If the condition is met, the current motion is immediately aborted, and the program continues with the subsequent lines. If the condition is not met, the motion will continue until the endpoint.

Sub-parameters:

DI: Input signal, supporting only DI1 to DI16.

Value: Input signal value, either True or False.

Example:

//Monitor DI during motion. If it is True, the motion is immediately aborted; otherwise, the motion continues until the endpoint.

MoveLJump RobotTarget,vel1,zone1,JumpData,Until DI1=True;

Note:

When using the Until parameter, the cornering zone setting will not take effect.

- **Out(DO,Output Value,Type,Type Value):** (Optional parameter) Output signals in parallel during execution.

A maximum of 2 signals can be output in parallel in a single motion instruction.

Sub-parameters:

DO: Output signals, supporting only DO1-DO16.

Output Value: Output signal value, either True or False.

Type: The type of signal output during motion. Different types of motion can use different Types.

Type Value: Type value.

Type	Meaning	Applicable Motion Types
Time(s)	Output signal based on time during motion: Type Value represents time, unit: seconds, range: -99 to 99. Type Value ≥ 0 means the signal will be output after the specified time from the start of motion. Type Value < 0 means the signal will be output a specified time before reaching the target point. When the Type Value exceeds the motion time range, the signal will be triggered only at the start and end points of that segment, and the specified time will be invalid.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Percentage(%)	Output signal based on path percentage during motion: Type Value represents the path percentage, indicating that the signal will be output when the motion reaches Type Value % of the total path. The range is from 0 to 100.	MoveL、MoveJ、MoveC、MoveAbsJ、MoveLJump
Distance(mm)	Output signal based on distance during motion: Type Value represents distance, unit: mm, range:	MoveL、MoveC、MoveLJump

	-1999 to 1999. Type Value ≥ 0 means the signal will be output after moving the specified distance from the starting point. Type Value < 0 means the signal will be output when the motion reaches the specified distance from the endpoint.	
--	--	--

Note:

The accuracy of motion I/O set within the cornering zone cannot be guaranteed.

When the specified Type Value exceeds the motion range, the signal will be triggered only at the start and end points of that segment.

Example:

//After 3 seconds of motion, DO1 outputs True. When there are 2 seconds remaining before the motion ends, DO2 outputs True.

```
MoveLJump RobotTarget,vel1,zone1,JumpData,Out(DO1,True,T,3),Out(DO2,True,T,-2);
```

//When the motion reaches 10% of the path, DO1 outputs True. When the motion reaches 100% of the path, DO2 outputs True.

```
MoveLJump RobotTarget,vel1,zone1,JumpData,Out(DO1,True,P,10),Out(DO2,True,P,100);
```

//After moving 50mm, DO1 outputs True. When there are 10mm remaining to the endpoint, DO2 outputs True.

```
MoveLJump RobotTarget,vel1,zone1,JumpData,Out(DO1,True,D,50),Out(DO2,True,D,-10);
```

7.3.6. MovePath

Description

Feedforward motion.

Feedforward (MovePath) refers to the control system processing the program instructions that follow the current instruction the robot is executing, in advance, during the robot's motion.

The introduction of the feedforward mechanism has the following benefits:

- It allows for the anticipation of the upcoming trajectory, enabling the planning of the optimal control strategy.

When performing feedforward motion, the following points must be noted:

- The target points in the feedforward instructions are represented by the TCP (Tool Center Point) in the base coordinate system.
- Feedforward motion can only be integrated with Cartesian motion, and only the parameters of the first feedforward instruction can be integrated.
- Feedforward motion does not support External.
- Feedforward motion does not support Offs and RelTool.

Format

MovePath RobotTarget, Vel, Zone, [Wobj], [Tool], [Acc];

Among them, the parameters enclosed in [] are optional and can be left unset.

Parameters

- **RobotTarget:** The target position described in Cartesian space. **Data Type:** RobotTarget
- **Vel:** Specifies the speed.
- **Zone:** Cornering zone.
- **Wobj:** (Optional parameter) The selected workpiece coordinate system, which can be one of wobj1 to wobj16.

When this parameter is not set, the workpiece coordinate system in the target pose RobotTarget is used.

For example, the MovePath command in the image below uses the wobj4 workpiece coordinate system.

```
18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);
19
1 MovePath targetpoint,vel1,zone1;
```

- **Tool:** (Optional parameter) The selected tool coordinate system, which can be one of tool1 to tool16.

When this parameter is not set, the tool coordinate system in the target pose RobotTarget is used.

For example, the MovePath command in the image below uses the tool2 tool coordinate system.

```
18 targetpoint : RobotTarget:=((-752.9393,20,-20,0,0,0),(0,0,0,0,0,0),(0,0,0,0,0,0),tool2,wobj4);  
19  
1 MovePath targetpoint,vel1,zone1;
```

- **Acc:** (Optional parameter) Specify acceleration.

When this parameter is not set, the default value is (10, 10, 10, 10, 10) .

7.3.7. AutoLJM

Description

Enable minimum path.

Format

AutoLJM(Enable Minimum Path);

Parameter

Enable Minimum Path: Whether to enable the minimum path. Data type: Bool.

Example

AutoLJM(True); //Enable minimum path.

AutoLJM(False); // Disable minimum path.

7.4. Mathematical Command

7.4.1. Sin

Description

Calculate sine operation, in radians.

Format

Return Value := Sin(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Sin(3.14/2);

7.4.2. Cos**Description**

Calculate cosine operation, in radians.

Format

Return Value := Cos(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Doubletype.

Example

Double1 := Cos(3.14/2);

7.4.3. Tan**Description**

Calculate tangent operation, in radians.

Format

Return Value := Tan(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Tan(3.14/2);

7.4.4. Asin**Description**

Calculate arcsine operation.

Format

Return Value := Asin(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Doubletype.

Example

Double1 := Asin(1*0.5);

7.4.5. Acos**Description**

Calculate arccosine operation.

Format

Return Value := Acos(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example


```
Double1 := Acos(1*0.5);
```

7.4.6. Atan

Description

Calculate arctangent operation.

Format

```
Return Value := Atan(<X>);
```

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

```
Double1 := Atan(1*0.5);
```

7.4.7. Exp

Description

Calculate the value of x raised to the power of e.

Format

```
Return Value := Exp(<X>);
```

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

```
Double1 := Exp(2/2);
```

7.4.8. Ln

Description

Calculate the natural logarithm of x (logarithm with base e).

Format

Return Value := Ln(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Ln(2/2);

7.4.9. Log10

Description

Calculate the logarithm of x with base 10.

Format

Return Value := Log10(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Log10(100*10);

7.4.10. Pow

Description

Calculate X raised to the power of Y.

Format

`<Var> := Pow(<X>,<Y>);`

Parameters

`<X>`: Mathematical expression for the base.

`<Y>`: Mathematical expression for the exponent.

Return value

Return Value: Double type.

Example

`Double1 := Pow(2,3);`

7.4.11. Sqrt

Description

Square root operation.

Format

`Return Value := Sqrt(<X>);`

Parameters

`<X>`: Mathematical expression, must be a positive number.

Return value

Return Value: Double type.

Example

`Double1 := Sqrt(4);`

7.4.12. Ceil

Description

Return the smallest integer value greater than or equal to X.

Format

Return Value := Ceil(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Ceil(4.1);

7.4.13. Floor

Description

Return the largest integer value less than or equal to X.

Format

Return Value := Floor(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Floor(-4.1);

7.4.14. Round

Description

Return the rounded value of X.

Format

Return Value := Round(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Round(-4.1);

Note

For a number in the form of a.5:

When a is positive:

If a is odd, rounding it will cause it to round up, i.e., the result will be a + 1.

Example: 3.5 => 4.

If a is even, rounding it will discard the 5, i.e., the result will remain a.

Example: 4.5 => 4.

When a is negative:

If a is odd, rounding it will cause it to round down, i.e., the result will be a - 1.

Example: -3.5 => -4.

If a is even, rounding it will discard the 5, i.e., the result will remain a.

Example: -4.5 => -4.

7.4.15. Min

Description

Return the minimum value between X and Y.

Format

Return Value := Min(<X>,<Y>);

Parameters

<X>: Mathematical expression.

<Y>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Min(-4.1,5);

7.4.16. Max

Description

Return the maximum value between X and Y.

Format

Return Value := Max(<X>,<Y>);

Parameters

<X>: Mathematical expression.

<Y>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Max(-4.1,5);

7.4.17. Abs**Description**

Return the absolute value of X.

Format

Return Value := Abs(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Double type.

Example

Double1 := Abs(-4.1);

7.4.18. Sign**Description**

Return the sign of X: 1 for positive numbers, -1 for negative numbers.

Format

Return Value := Sign(<X>);

Parameters

<X>: Mathematical expression.

Return value

Return Value: Int type.

Example

```
Int1 := Sign(-4.1);
```

7.5. IO Command

7.5.1. SetDO

Description

Set the output value of a digital signal.

Format

```
SetDO(OutputSignal,OutputValue);
```

Parameters

OutputSignal: Output signal index, data type is Int.

OutputValue: Output value, data type is Bool.

Example

```
SetDO(int1,True);
```

7.5.2. GetDI

Description

Get the input value of a digital signal.

Format

```
Return Value := GetDI(Input Signal);
```

Parameters

Input Signal: Input signal index value, data type is Int.

Return value

Return value type: Bool.

Example

```
Bool1 := GetDI(int1);
```


7.5.3. PulseDO

Description

Generate a pulse for a DO signal.

Format

PulseDO(Output Signal,Pulse Duration);

Parameters

OutputSignal: Output signal index, data type is Int.

PulseDuration: Pulse duration (s), data type is Double.

Example

PulseDO(int1,double1);

7.5.4. Group

Description

Configure input and output group signals.

Description

A signal group by default contains 8 signals. For example, OG1 contains DO1-8, OG2 contains DO9-16, and so on. After using the Group instruction, the output can be redefined. The number of signals in the new group can be less than 8. However, note that this is only valid within the current program.

Format

Group(Configure Signal Group,n0, n1,n2...,n7);

Parameters

Configure Signal Group: The input or output group to be redefined, ranging from OG1 to OG8, IG1 to IG8, data type is String.

n0, n1,n2...,n7: Signal indices. Data Type:Int or Word. Multiple selections are allowed, with a maximum of 8 and a minimum of 2.

Example

```
Group('OG1',8,7,6,5,4,3,2,1);  
Group('IG1',1,3,5,7,9,11,13,15);
```

7.5.5. SetOG**Description**

Set a group of digital output signals.

Format

```
SetOG(Output signal Group, Signal Status);
```

Parameters

Output signal Group: Output signal group, ranging from OG1 to OG8. By default, each group contains 8 signals, data type is String.

Signal Status: The status of a signal group. Data Type: Word. The value of it is an integer between <0-255> which exactly corresponds to an 8-bit binary number. Each bit, from right to left, represents the signal status from low to high.

Example

```
Group('OG4',1,3,5,7,9,11,13,15);  
SetOG('OG4',255);
```

7.5.6. GetIG**Description**

Read a group of digital input signals.

Format

```
Return Value := GetIG(Input Signal Group);
```

Parameters

Input Signal Group: Output signal group, ranging from OG1 to OG8. By default, each group contains 8 signals, data type is String.

Return value

Return Value: Word type variables. The value of it is an integer between <0-255> , which exactly corresponds to an 8-bit binary number. Each bit, from right to left, represents the signal status from low to high.

Example

```
Group('IG4',1,3,5,7,9,11,13,15);  
  
value1 := GetIG('IG4');
```

7.6. Program Command

7.6.1. Call

Description

Call a subroutine.

Format

```
Call('Name');
```

Parameters

Name: Program path.

Example

```
Call('Test/Test1');
```

Note:

Circular program calls are not allowed.

7.7. SocketTCP Command

7.7.1. SocketCreate

Description

Establish a Socket connection.

Format

Return Value := SocketCreate(IP Address,Port,Socket Name);

Parameters

IPAddress: Server IP address, data type is String.

Port: Server port number, data type is Int, with a valid range from 1025 to 49151.

SocketName: Socket name, valid values are Client1-Client4, Server1-Server4.

Return value

Return value type: Bool

Example

```
bool1 := SocketCreate('192.168.100.103',int1,'Client1');
```

7.7.2. SocketClose

Description

Close the Socket.

Format

SocketClose(Socket Name);

Parameters

Socket Name: Socketname, valid values are Client1-Client4, Server1-Server4.

Example

```
SocketClose('Client1');
```

7.7.3. SocketSendString

Description

Send a string externally via the Socket.

Format

SocketSendString(Content,Socket Name);

Parameters

Content: Content, data type is String.

SocketName: Socket name, valid values are Client1-Client4, Server1-Server4.

Example

```
SocketSendString(string1,'Client1');
```

7.7.4. SocketReadString

Description

Read a string from the Socket and return it.

Format

Return Value := SocketReadString(Socket Name);

Parameters

SocketName: Socket name, valid values are Client1-Client4, Server1-Server4.

Return value

Return value type: String type.

Example

```
string1 := SocketReadString('Client1');
```

7.7.5. SocketStatus

Description

Return the Socket status.

Format

Return Value := SocketStatus(Socket Name);

Parameters

SocketName: Socket name, valid values are Client1-Client4, Server1-Server4.

Return value

Return value type: Int type.

0: Not connected, 1: Connected, 2: Server created.

Example

```
int1 := SocketStatus('Client1');
```

7.8. String Command

7.8.1. StrConcat

Description

Concatenate two strings and return the result.

Format

```
Return Value := StrConcat(String1,String2);
```

Parameters

String1: String 1, data type is String.

String2: String 2, data type is String.

Return value

Return value type: String type.

Example

```
string1 := StrConcat(string2,string3);
```

7.8.2. StrDelete

Description

Delete the character at the specified position and return the result.

Format

```
Return Value := StrDelete(String,Length,Position);
```

Parameters

String: string. The data type is String type.

Length: length. The data type is Int type.

Position: position. The data type is Int type.

Return value

Return value type: String.

Example

```
string1 := StrDelete(string2,int1,int2);
```

7.8.3. StrFind

Description

Find the starting index of the specified substring in the string.

Format

Return Value := StrFind(Main String,Search String);

Parameters

MainString: Main string. The data type is String type.

SearchString: Search string. The data type is String type.

Return value

Return value type: Int.

Example

```
int1 := StrFind(string1,string2);
```

7.8.4. StrInsert

Description

Insert another string at a specific position in the string.

Format

Return Value := StrInsert(Main String,Insert String,Insert Position);

Parameters

MainString: Main string. The data type is String type.

InsertString: The string to be inserted, data type is String.

InsertPosition: The position at which to insert, data type is Int.

Return value

Return value type: String

Example

```
string1 := StrInsert(string2,string3,int1);
```

7.8.5. StrLeft

Description

Extract the leftmost characters of a string with a specified length.

Format

Return Value := StrLeft(String,Length);

Parameters

String: string. The data type is String type.

Length: length. The data type is Int type.

Return value

Return value type: String.

Example

```
string1 := StrLeft(string2,int1);
```

7.8.6. StrLen

Description

Get the length of the string.

Format

Return Value := StrLen(String);

Parameters

String: string. The data type is String type.

Return value

Return value type: Int.

Example

```
int1 := StrLen(string1);
```

7.8.7. StrMid

Description

Extract a substring from the middle of the string.

Format

Return Value := StrMid(String,Length,Position);

Parameters

String: string. The data type is String type.

Length: length. The data type is Int type.

Position: position. The data type is Int type

Return value

Return value type: String.

Example

```
string1 := StrMid(string2,int1,int2);
```

7.8.8. StrReplace

Description

Replace a specified number of characters at a specific position in the string.

Format

Return Value := StrReplace(Main String,Replace String,Length,Position);

Parameters

MainString: Main string. The data type is String type.

ReplaceString: Replacement string, data type is String.

Length: length. The data type is Int type.

Position: position. The data type is Int type

Return value

Return value type: String.

Example

string1 := StrReplace(string2,string3,int1,int2);

7.8.9. StrRight

Description

Extract the rightmost characters of a string with a specified length.

Format

Return Value := StrRight(String,Length);

Parameters

String: string. The data type is String type.

Length: length. The data type is Int type.

Return value

Return value type: String.

Example

string1 := StrRight(string2,int1);

7.8.10. StrToDouble

Description

Convert the string to a Double data type.

Format

Return Value := StrToDouble(String);

Parameters

String: string. The data type is String type.

Return value

Return Value: Double type.

Example

```
double1 := StrToDouble(string1);
```

7.8.11. StrToInt

Description

Convert the string to an Int data type.

Format

Return Value := StrToInt(String);

Parameters

String: string. The data type is String type.

Return value

Return value type: Int.

Example

```
int1 := StrToInt(string1);
```

7.8.12. StrToWord

Description

Convert the string to a Word data type.

Format

Return Value := StrToWord(String);

Parameters

String: string. The data type is String type.

Return value

Return value type: Word type.

Example

```
word1 := StrToWord(string1);
```

7.8.13. StrToFloat

Description

Convert the string to a Float data type.

Format

Return Value := StrToFloat(String);

Parameters

String: string. The data type is String type.

Return value

Return value type: Float type.

Example

```
float1 := StrToFloat(string1);
```

7.8.14. DoubleToStr

Description

Convert Double data to a string.

Format

Return Value := DoubleToStr(Double);

Parameters

Double: Data type is Double.

Return value

Return value type: String.

Example

```
string1 := DoubleToStr(double1);
```

7.8.15. DoubleToInt

Description

Convert Double data to an Int data type.

Format

Return Value := DoubleToInt(Double);

Parameters

Double: Data type is Double.

Return value

Return value type: Int.

Example

```
Int1 := DoubleToInt(double1);
```

7.8.16. DoubleToWord

Description

Convert Double data to a Word data type.

Format

Return Value := DoubleToWord(Double);

Parameters

Double: Data type is Double.

Return value

Return value type: Word.

Example

```
word1 := DoubleToWord(double1);
```

7.8.17. DoubleToFloat

Description

Convert Double data to a Float data type.

Format

Return Value := DoubleToFloat(Double);

Parameters

Double: Data type is Double.

Return value

Return value type: Float type.

Example

```
float1 := DoubleToFloat(double1);
```

7.9. Pallet Command

7.9.1. SetPallet

Description

Sets the global pallet variable.

Description

Creates a parallelogram using three points, Point1, Point2, and Point3, as the "boundary points" of the pallet. Point1 specifies the first point on the pallet, the line connecting Point1 and Point2 defines the row direction, and the line connecting Point1 and Point3 defines the column direction. The command works by creating the pallet boundary based on the three input points and then generating the pallet model according to the number of rows, columns, layers, and layer height. After setup, the information for any position on the pallet can be

obtained using just the row, column, and layer indices.

Format

```
SetPallet(Pallet Number,Point1,Point2,Point3,Rows,Columns,Layer Count,Layer Height);
```

Parameters

Pallet Number: Pallet ID, data type Int, range 1–64.

Point1: Pallet point 1, data type RobotTarget.

Point2: Pallet point 2, data type RobotTarget.

Point3: Pallet point 3, data type RobotTarget.

Rows: Number of rows, data type Int, range 1–1000.

Columns: Number of columns, data type Int, range 1–1000.

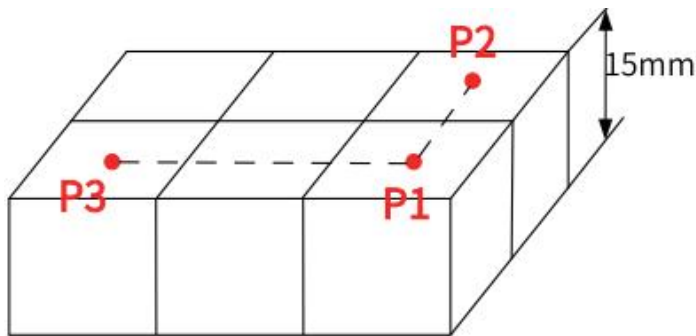
Layer Count: Number of layers, data type Int, range 1–1000.

Layer Height: Layer height, data type Double, unit: mm.

Example

```
//Set a pallet with 2 rows, 3 columns, 1 layer, and a layer height of 15 mm.
```

```
SetPallet(Pallet1,p1,p2,p3,2,3,1,15);
```



Note:

The maximum pallet capacity is 1000.

7.9.2. SetPallet4

Description

Sets a pallet using four points. Unlike SetPallet, this command determines the four edges of the

pallet through four points to obtain array points. It is recommended for scenarios requiring high placement accuracy or when the pallet may be deformed.

Format

```
SetPallet4(PalletNumber,Point1,Point2,Point3,Point4,Rows,Columns,Layer Count,Layer Height);
```

Parameters

PalletNumber: pallet index, data type Int, valid range 1–64.

Point1: pallet point 1, data type RobotTarget.

Point2: pallet point 2, data type RobotTarget.

Point3: pallet point 3, data type RobotTarget.

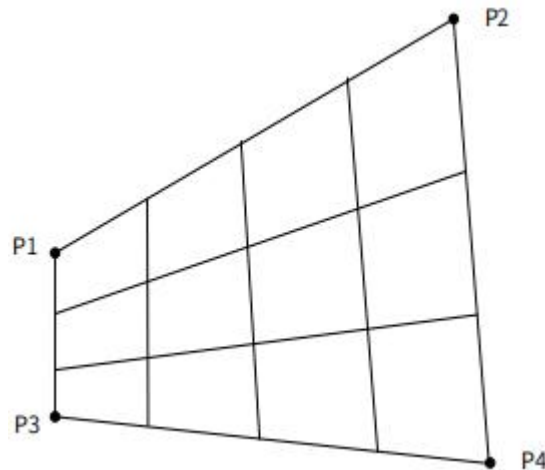
Point4: pallet point 4, data type RobotTarget.

Rows: number of rows, data type Int, valid range 1–1000.

Columns: number of columns, data type Int, valid range 1–1000.

Layer Count: number of layers, data type Int, valid range 1–1000.

Layer Height: layer height, data type Double, unit: mm.

**Return value**

None

Example

```
//Set a pallet Pallet2 with 2 rows, 3 columns, 4 layers, and a layer height of 20mm.
```

```
SetPallet(Pallet2,pos1,pos2,pos3,pos4,2,3,4,20);
```


7.9.3. Pallet

Description

Gets a point on the pallet. Can be used together with the SetPallet or SetPallet4 commands.

Format

Return Value := Pallet(Pallet Number, Rows, Columns, Layer Count);

Parameters

Pallet Number: pallet index, data type Int, valid range 1–64.

Rows: row number, data type Int, valid range 1–1000.

Columns: column number, data type Int, valid range 1–1000.

Layer Count: layer number, data type Int, valid range 1–1000.

Return value

Return Value: RobotTarget type.

Example

```
//Get the point at row 1, column 2, layer 1 on Pallet1.
```

```
RobotTarget1 := Pallet(Pallet1,1,2,1);
```

7.9.4. PalletSequence

Description

Retrieves a point from the pallet. Can be used together with SetPallet and SetPallet4.

Format

Return Value := PalletSequence(Pallet Number, Location Number);

Parameters

Pallet Number: pallet index, data type Int, valid range 1–64.

Location Number: location index, data type Int.

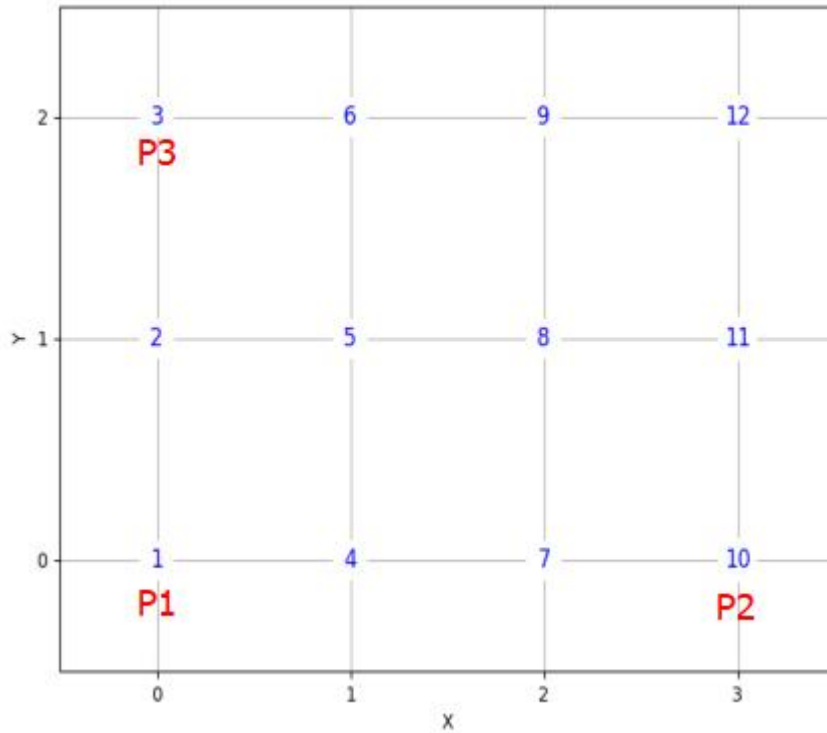
Return value

Return Value: RobotTarget type.

Example

```
//Retrieve the point with location index 1 from pallet Pallet1.
```

```
RobotTarget1 := PalletSequence(Pallet1, 1);
```



7.10. Modbus Command

7.10.1. GetModbusCoilToBool (Deprecated)

Description

Reads the coil value from a Modbus slave and stores it in a Bool type variable.

Format

```
Return Value := GetModbusCoilToBool(Starting Address);
```

Parameters

Starting Address: Coil starting address, data type: Int, valid range: 1000–65535.

Return value

Return Value: Bool type.

Example

```
bool1 := GetModbusCoilToBool(int1);
```

Note

This command has been marked as “deprecated.” It was originally used in Gorilla Studio control system version 1.17. It is still compatible in later versions but is no longer maintained. Users are recommended to use alternative commands to ensure compatibility and long-term support.

7.10.2. SetModbusCoilByBool (Deprecated)**Description**

Sets the Modbus slave coil value based on the value of a Bool variable.

Format

```
SetModbusCoilByBool(Starting Address,Set Value);
```

Parameters

Starting Address: Starting address of the coil, data type Int, range 1000–65535.

Set Value: Value to set, data type Bool.

Example

```
SetModbusCoilByBool(int1,bool1);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.3. GetModbusRegToWord (Deprecated)**Description**

Reads one value from a Modbus register and forcibly converts it into a Word-type variable according to the specified byte order.

Format

Return Value := GetModbusRegToWord(Starting Address);

Parameters

Starting Address: Starting address of the register, data type Int, range 1000–65535.

Return value

Return Value: Word type.

Example

Word := GetModbusRegToWord(int1);

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.4. SetModbusRegByWord (Deprecated)

Description

Based on the value of the Word variable, forcibly convert it into a Modbus slave register value, following the specified byte order.

Format

SetModbusRegByWord(Starting Address,Set Value);

Parameters

Starting Address: Starting address of the register, data type Int, range 1000–65535.

Set Value: Value to set, data type Word.

Example

SetModbusRegByWord(int1,word1);

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.5. GetModbusRegToInt (Deprecated)

Description

Reads two register values from the Modbus registers and forcibly converts them into an Int variable according to the specified byte order.

Format

Return Value := GetModbusRegToInt(Starting Address);

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Return value

Return Value: Int type.

Example

Int2 := GetModbusRegToInt(int1);

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.6. SetModbusRegByInt (Deprecated)

Description

Based on the value of the Int variable, forcibly convert it into 2 Modbus slave register values, following the specified byte order.

Format

SetModbusRegByInt(Starting Address,Set Value);

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Set Value: Value to set, data type Int.

Example

```
SetModbusRegByInt(int1,int2);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.7. GetModbusRegToDouble (Deprecated)

Description

Reads 4 register values from Modbus register and forcibly converts them into a Double variable according to the specified byte order.

Format

Return Value := GetModbusRegToDouble(Starting Address);

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Return value

Return Value: Double type.

Example

```
double1 := GetModbusRegToDouble(int1);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.8. SetModbusRegByDouble (Deprecated)

Description

Based on the value of the Double variable, forcibly convert it into 4 Modbus slave register values, following the specified byte order.

Format

SetModbusRegByDouble(Starting Address,Set Value);

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Set Value: Value to set, data type Double.

Example

```
SetModbusRegByDouble(int1,double1);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.9. GetModbusRegToFloat (Deprecated)

Description

Reads 2 registers from the Modbus register and forcibly convert them into a Float type variable according to the specified byte order.

Format

Return Value := GetModbusRegToFloat(Starting Address);

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Return value

Return Value: Float type.

Example

```
float1 := GetModbusRegToFloat(int1);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.10. SetModbusRegByFloat (Deprecated)**Description**

Converts the value of the Float variable into 2 Modbus registers according to the specified byte order by force.

Format

```
SetModbusRegByFloat(Starting Address,Set Value);
```

Parameters

Starting Address: Starting address of the registers, data type Int, range 1000 to 65535.

Set Value: Value to set, data type Float.

Example

```
SetModbusRegByFloat(int1,float1);
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.11. ModbusTcpStatus (Deprecated)**Description**

Get the Modbus connection status.

Format

Return Value := ModbusTcpStatus();

Return value

Return Value: Int type.

0: Not connected. 1: Connected. 2: Server created

Example

```
int1 := ModbusTcpStatus();
```

Note

This command is marked as deprecated. It was originally used in Gorilla Studio Control System version 1.17. It is still compatible in later versions but is no longer maintained. Users are advised to use alternative commands to ensure compatibility and long-term use.

7.10.12. ModbusStatus

Description

Get Modbus status

Format

Return Value := ModbusStatus(Modbus Number);

Parameters

Modbus Number: Modbus ID (1–4), where 1 is the slave, and 2, 3, 4 correspond to three different masters. Data type: Int.

Return value

Return Value: Int type.

0: Not connected. 1: Connected. 2: Modbus activated

Example

```
int1 := ModbusStatus(2);
```

7.10.13. GetModbusCoil

Description

Get coil values from the Modbus slave and store them in a Bool type array.

Format

Return Value := GetModbusCoil(Starting Address,Number of Coils Read,Modbus Number);

Parameters

Starting Address: Starting address of the coil, data type Int, range 1000–65535.

Number of Coils Read: Number of coils to read (1–1024), equal to the length of the Bool type variable array to store the values, data type Int (must be an integer constant).

Modbus Number: Modbus ID (1–4), where 1 is the slave, and 2, 3, 4 correspond to three different masters, data type Int.

Return value

Return Value: Bool type array variable.

Example

```
lb2 := GetModbusCoil(1000,100,2);
```

7.10.14. SetModbusCoil

Description

Sets the coil values of the Modbus slave according to the values of a Bool type variable array.

Format

SetModbusCoil(Starting Address,Number of Coils Written,Set Value,Modbus Number);

Parameters

Starting Address: Starting address of the coil, data type Int, range 1000–65535.

Number of Coils Written: Number of coils to write (1–1024), equal to the length of the Bool type variable array, data type Int (must be an integer constant).

Set Value: Set the Modbus slave coil values according to this variable array, data type Bool[[]].

Modbus Number: Modbus number (1–4), 1 for slave, 2, 3, 4 correspond to three different masters, data type Int.

Example

```
SetModbusCoil(1000,100,1b1,2);
```

7.10.15. GetModbusReg**Description**

Reads Number of Registers Read data items from Modbus registers according to the specified Data Type for Reading and save them into an array variable of the corresponding type.

Format

Return Value := GetModbusReg(Starting Address,Number of Registers Read,Data Type for Reading,Modbus Number);

Parameters

Starting Address: Starting address of the register, data type Int, range 1000–65535.

Number of Registers Read: Number of registers to read (1–1024), equal to the length of the return value variable array, data type Int (must be an integer constant).

Data Type for Reading: Data type for reading. 1: Word[], 2: Int[], 3: Float[], 4: Double[].

Modbus Number: Modbus number (1–4), 1 for slave, 2, 3, 4 correspond to three different masters, data type Int.

Return value

Return Value: Word[],Int[],Float[],Double[]。

Example

```
global_int := GetModbusReg(1000,10,2,1);
```

```
global_double := GetModbusReg(1000,100,4,2);
```

```
global_float := GetModbusReg(1000,1000,3,3);
```

```
global_word := GetModbusReg(1000,12,1,4);
```

7.10.16. SetModbusReg**Description**

Writes the values in Set Value into the specified register addresses according to the data type specified by Data Type for Writing.

Format

SetModbusReg(Starting Address,Number of data Written,Set Value,Data Type for Writing,Modbus Number);

Parameters

Starting Address: Starting address of the register, data type Int, range 1000–65535.

Number of data Written: Number of data items to write (1–1024), equal to the length of the variable array, data type Int (must be an integer constant).

Set Value: Values to write, data type Word[], Int[], Float[], or Double[].

Data Type for Writing: Data type for writing, data type Int (must be an integer constant).

1:Word[],2:Int[],3:Float[],4:Double[].

Modbus Number: Modbus number (1–4), 1 for slave, 2, 3, 4 correspond to three different masters, data type Int.

Example

```
SetModbusReg(1000,10,global_word,1,1);
```

```
SetModbusReg(1000,100,global_int,2,1);
```

```
SetModbusReg(1000,2,global_float,3,1);
```

```
SetModbusReg(1000,12,global_double,4,1);
```

7.11. Serial Port Command

7.11.1. ComOpen

Description

Opens the controller's serial port. Once ComOpen executes successfully, the port will remain open until it is closed using the ComClose command or manually by modifying its configuration. Before using this command, the serial port parameters must be configured.

Note: Using a serial port number that does not exist on the controller will trigger a program

exception and cause the program to stop running.

Format

Return Value := ComOpen(Serial Port Number);

Parameters

Serial Port Number: The serial port number (the controller only has one usable port, which is COM3), data type: String.

Return value

Return Value: Bool type. Returns True if the serial port is successfully opened; otherwise, returns False.

Example

```
b1 := ComOpen('COM3');
```

7.11.2. ComClose

Description

Closes an opened serial port.

Format

ComClose(Serial Port Number);

Parameters

Serial Port Number: The serial port number (the controller only has one usable port, which is COM3), data type: String.

Example

```
ComClose('COM3');
```

7.11.3. ComSendString

Description

Sends a string to the specified serial port.

Format

```
ComSendString(Serial Port Number,String);
```

Parameters

Serial Port Number: The serial port number (the controller only has one usable port, which is COM3), data type: String.

String: The string to be sent, data type: String.

Example

```
ComSendString('COM3',str1);
```

7.11.4. ComReadString

Description

Reads characters from the specified serial port. Each execution of this command clears the receive buffer.

Format

```
Return Value := ComReadString(Serial Port Number);
```

Parameters

Serial Port Number: The serial port number (the controller only has one usable port, which is COM3), data type: String.

Return value

String type. The string read; returns an empty string if no data is available.

Example

```
str1 := ComReadString('COM3');
```

7.11.5. ComStatus

Description

Gets the status of the specified serial port.

Format

```
Return Value := ComStatus(Serial Port Number);
```

Parameters

Serial Port Number: The serial port number (the controller only has one usable port, which is COM3), data type: String.

Return value

Return Value: Bool type. Returns True if the serial port is open, otherwise False.

Example

```
b1 := ComStatus('COM3');
```

7.12. Tracking Command

7.12.1. ConveyorVision

Description

Enables or disables the vision port of the conveyor. When the conveyor vision port is enabled, the controller automatically stores objects detected by vision into a queue without requiring the user to process the vision coordinates.

Description

- When objects are present, communication data format:

@X1,Y1,Z1,A1,T1,X2,Y2,Z2,A2,T2.....\r

X1: X coordinate of object 1

Y1: Y coordinate of object 1

Z1: Z coordinate of object 1

A1: Rz coordinate of object 1

T1: Type of object 1

X2: X coordinate of object 2

Y2: Y coordinate of object 2

Z2: Z coordinate of object 2

A2: Rz coordinate of object 2

T2: Type of object 2

- If no object is detected: NG

Note: When there are objects, the data packet starts with @ and ends with \r.

Format

ConveyorVision(Conveyor Belt ID,Turn On Conveyor Belt Vision,Socket Name);

Parameters

Conveyor Belt ID: Conveyor number, valid range 1–4.

Turn On Conveyor Belt Vision: Whether to enable the conveyor vision port, data type Bool.

True: Enable the vision port of the specified conveyor. False: Disable the vision port of the specified conveyor

Socket Name: Socket name, valid values: Client1–Client4, Server1–Server4

Example

// Enable vision port of conveyor 1, controller as client, vision as server

ConveyorVision(1, True, 'Client1');

7.12.2. ConveyorSensor

Description

Enable or disable the conveyor belt's sensor detection. When sensor detection is enabled, the controller will automatically store the objects detected by the sensor into a queue, without requiring the user to process them.

Format

ConveyorSensor(Conveyor Belt ID,Turn On Conveyor Belt Sensor Detection);

Parameters

Conveyor Belt ID: Conveyor belt number, valid range: 1 to 4.

Turn On Conveyor Belt Sensor Detection: Whether to enable conveyor belt sensor detection, data type: Bool.

Example


```
ConveyorSensor(1,True);
```

7.12.3. GetConveyorObject

Description

Query whether there is an object of the specified type in the pick-up area (between the working upper bound and the grasping lower bound) from the conveyor belt object queue. If there is, it returns True; otherwise, it returns False. Objects detected by vision or sensors are automatically stored in the robot controller's queue. The controller tracks the positions of these objects in real time. When an object enters the pick-up area, it can be queried by GetConveyorObject and return True.

Format

```
Return Value := GetConveyorObject(Conveyor Belt ID,Object Type ID);
```

Parameters

Conveyor Belt ID: Conveyor belt number, valid range: 1 to 4.

Object Type ID: Object type number, valid range: 1 to 15.

Return value

Return Value: Bool. Returns True if an object of the specified type exists in the pick-up area, otherwise returns False.

Example

```
// Query whether there is an object of type 2 in the pick-up area of conveyor belt 1  
  
bool1 := GetConveyorObject(1,2);
```

7.12.4. TrackConveyor

Description

Switch the motion reference coordinate system to the object coordinate system on the conveyor belt. After switching the motion reference to the object coordinate system on the conveyor belt, the description of trajectories and points is relative to the object coordinate system.

Format

TrackConveyor(Conveyor Belt ID);

Parameters

Conveyor Belt ID: Conveyor belt number, valid values are 1 to 4

Example

TrackConveyor(1);

Note

After executing TrackConveyor, the robot will continuously move in sync with the conveyor belt until TrackBase is executed. If TrackBase is not executed in time, the robot will follow the conveyor belt out of bounds. Therefore, logically, TrackConveyor and TrackBase must always be used in pairs, and the time interval between these two commands must not exceed the time it takes for the conveyor belt to move to the boundary.

7.12.5. TrackBase

Description

Switch the motion reference coordinate system to static.

Format

TrackBase;

Example

TrackBase;

7.13. Profinet Command

7.13.1. GetProfinetToWord

Description

Reads 1 Word from Profinet and forcibly converts it to a Word-type variable according to the specified byte storage format.

Format

Return Value := GetProfinetToWord(Register Address);

Parameters

Register Address: Register address, data type Int, value range 0–63.

Return value

Return Value: Word type.

Example

```
Word := GetProfinetToWord(int1);
```

7.13.2. SetProfinetByWord

Description

Converts a Word-type variable to 1 Word on Profinet according to the specified byte storage format.

Format

```
SetProfinetByWord(Register Address,Set Value);
```

Parameters

Register Address: Register address, data type Int, value range 0–63.

Set Value: Value to set, data type Word.

Example

```
SetProfinetByWord(int1,word1);
```

7.13.3. GetProfinetToInt

Description

Reads 2 Words from Profinet and forcibly converts them into an Int-type variable according to the specified byte storage format.

Format

```
Return Value := GetProfinetToInt(Register Address);
```

Parameters

Register Address: Register address, data type Int, value range 0–63.

Return value

Return Value: Int type.

Example

```
Int := GetProfinetToInt(int1);
```

7.13.4. SetProfinetByInt

Description

Converts an Int-type variable into 2 Words on Profinet according to the specified byte storage format.

Format

```
SetProfinetByInt(Register Address,Set Value);
```

Parameters

Register Address: Register address, data type Int, value range 0–63.

Set Value: Value to set, data type Int.

Example

```
SetProfinetByInt(int1,int2);
```

7.13.5. GetProfinetToDouble

Description

Reads 4 Words from Profinet and forcibly converts them into a Double-type variable according to the specified byte storage format.

Format

```
Return Value := GetProfinetToDouble(Register Address);
```

Parameters

Register Address: Register address, data type Int, value range 0–63.

Return value

Return Value: Double type.

Example

```
Double := GetProfinetToDouble(int1);
```

7.13.6. SetProfinetByDouble

Description

Converts a Double-type variable into 4 Words on Profinet according to the specified byte storage format.

Format

```
SetProfinetByDouble(Register Address,Set Value);
```

Parameters

Register Address: Register address, data type Int, value range 0–63.

Set Value: Value to set, data type Double.

Example

```
SetProfinetByDouble(int1,double);
```

7.13.7. ProfinetStatus

Description

Gets the status of the Profinet slave.

Format

```
Return Value := ProfinetStatus();
```

Return value

Return Value: Int type.

0: Not connected, 1: Connected, 2: Profinet slave activated

Example

```
int1 := ProfinetStatus();
```

7.14. Fly Camera Command

7.14.1. InitialFly

Description

Initializes the fly camera function.

Format

```
InitialFly();
```

Example

```
InitialFly();
```

7.14.2. SetFly

Description

Set when to trigger the fly camera function. The parameters are taken from the fly camera settings interface.

Format

```
SetFly(Flying Shooting ID);
```

Parameters

Flying Shooting ID: Fly camera index, data type Int, value range 1–16.

Example

```
SetFly(i1);
```

7.14.3. StartFly

Description

Fly camera start motion flag. Execute this command before reaching the fly camera end point.

Format

```
StartFly();
```

Example

```
StartFly();
```

7.14.4. StopFly**Description**

Fly camera stop motion flag. Execute this command after reaching the fly camera end point.

Format

```
StopFly();
```

Example

```
StopFly();
```

7.14.5. CheckFly**Description**

Checks whether the fly camera motion with the specified index was successful.

Format

```
Return Value := CheckFly(Flying Shooting ID);
```

Parameters

Flying Shooting ID: Fly camera index, data type Int, value range 1–16.

Return value

Return Value: Bool type. Returns True if the fly camera executes successfully, otherwise False.

Example

```
b1 := CheckFly(i1);
```

7.14.6. GetFlyRobotPose**Description**

Gets the robot's position at the moment of fly camera.

Format

Return Value := GetFlyRobotPose(Flying Shooting ID);

Parameters

Flying Shooting ID: Fly camera index, data type Int, value range 1–16.

Return value

Return Value: Pose type. The robot's position when the fly camera is executed.

Example

```
pose1 := GetFlyRobotPose(i1);
```

7.14.7. CalculationFlyResult

Description

Calculates the final placement position for the fly camera.

Format

Return Value := CalculationFlyResult(Flying Shooting ID, Tool Index, Robot Position, Vision Coordinates, Placement Point);

Parameters

Flying Shooting ID: Fly camera index, data type Int, value range 1–16.

Tool Index: Tool index (the tool must be used during the placement motion), data type Int, value range 2–16.

Robot Position: Robot position at the time of fly camera, data type Pose.

Vision Coordinates: Coordinates of the vision marker at the time of fly camera, data type Pose.

Placement Point: Placement point information from fly camera calibration, data type RobotTarget.

Return value

Return Value: RobotTarget type. The calculated final placement position.

Example

Targetpose := CalculationFlyResult(1,2,Robotpose,Visionpose,Placepose);

7.15. Other Commands

7.15.1. GetCartesianPose

Description

Get the robot's Cartesian space pose.

Format

Return Value := GetCartesianPose(Parameter);

Parameters

Parameter: Parameter, data type Int.

1: Returns the TCP pose under the currently selected tool coordinate system and work object coordinate system in the toolbar.

2: Returns the pose of the flange end relative to the base coordinate system.

Return Value

Return Value: Pose type.

Example

Pose := GetCartesianPose(1);

7.15.2. GetAxisPosition

Description

Get the robot joint-space pose.

Format

Return Value := GetAxisPosition(Parameter);

Parameters

Parameter: Parameter, data type Int.

1: Returns the robot axis positions.

2: Returns the external axis positions.

Return value

Return Value: AxisPos type.

Example

```
AxisPos := GetAxisPosition(1);
```

7.15.3. GetRobotTarget

Description

Get the robot pose under the specified tool coordinate system and work object coordinate system.

Format

```
Return Value := GetRobotTarget(Tool ID,Wobj ID);
```

Parameters

Tool ID: Tool coordinate system ID, data type Int.

Wobj ID: Work object coordinate system ID, data type Int.

Return value

Return Value: RobotTarget type. Returns the robot's current position, orientation, axis configuration data, and external axis information.

Example

```
RobotTarget1 := GetRobotTarget(i1,i2);
```

7.15.4. GetRobotStatus

Description

Get the current robot status.

Format

```
Return Value := GetRobotStatus();
```

Return value

Return Value: Word type.

Robot Status Code	Robot Status
0	Robot not configured
1	Robot not enabled
2	Robot enabled
3	Robot homing
4	Robot in motion
5	Robot stopping
6	Robot fault
7	Robot jogging
9	Robot emergency stop
100	System not connected

Example

```
word1 := GetRobotStatus();
```

7.15.5. GetRunStatus**Description**

For multitask functionality, get the main task running status.

Format

```
Task Status:=GetRunStatus();
```

Parameters

No parameters.

Return value

Task Status: Main task running status, data type Word.

0: Stopped

1: Running

2: Paused

3: Single-step running

Example

```
Status:=GetRunStatus();
```

7.15.6. ZeroCalibration

Description

For DELTA robot four-axis homing, applicable to BAT_360 (fourth axis is stepper motor) model.

Format

```
ZeroCalibration(Signal Index,Zero Return Method,Zero Return Speed,Zero Return Direction);
```

Parameters

Signal Index: Input signal index, data type Int.

Zero Return Method: Homing method, data type Int.

0: Signal triggers homing in positive direction;

1: Signal triggers homing in negative direction;

2: Torque homing;

3: User homing.

Zero Return Speed: Homing speed, data type Double.

Zero Return Direction: Homing direction, data type Bool.

Example

```
ZeroCalibration(1,3,10,True);
```

7.15.7. UToEuler

Description

Convert the taught pose with the U axis enabled to the pose representation with the U axis

disabled.

Format

Return Value := UToEuler(Parameter);

Parameters

Parameter: Taught pose with U axis enabled, data type RobotTarget.

Return value

Return Value: Pose representation with U axis disabled, data type RobotTarget.

Example

ReturnPose1 := UToEuler(Upose2);

7.15.8. EulerToU

Description

Convert the taught pose with the U axis disabled to the pose representation with the U axis enabled.

Format

Return Value := EulerToU(Parameter);

Parameters

Parameter: Pose representation with U axis disabled, data type RobotTarget.

Return value

Return Value: Pose representation with U axis enabled, data type RobotTarget.

Example

ReturnPose1 := EulerToU(Pose2);

7.15.9. BitSet

Description

Sets a specific bit of Int or Word type data to 1 or 0, with bit positions starting from 0.

Format

Return Value := BitSet(Data,Position,Modify Value);

Parameters

Data: The integer data to operate on, data type Int.

Position: The position to operate on. For Word type data, 0–15; for Int type data, 0–31. Data type Word.

Modify Value: The value to modify, data type Bool.

Return value

Return Value: The value of Data after the specified bit has been modified. Data type: Int.

Example

```
value1 := BitSet(i,0,b);
```

7.15.10. BitCheck

Description

Used to check whether a specific bit in a defined Int or Word type data is 1. Returns True if it is 1; otherwise, returns False.

Format

Return Value := BitCheck(Data,Position);

Parameters

Data: The integer data to be operated on. Data type: Int.

Position: The position to be operated on. For Word type data, valid positions are 0–15; for Int type data, valid positions are 0–31. Data type: Word.

Return value

Return Value: The result of the check. Data type: Bool.

Example

```
b2 := BitCheck(value1,0);
```

7.15.11. PoseMultiplication

Description

Used to calculate the product of two poses.

Format

Return Value := PoseMultiplication(Pose1,Pose2);

Parameters

Pose1: The first pose. Data type: Pose.

Pose2: The second pose. Data type: Pose.

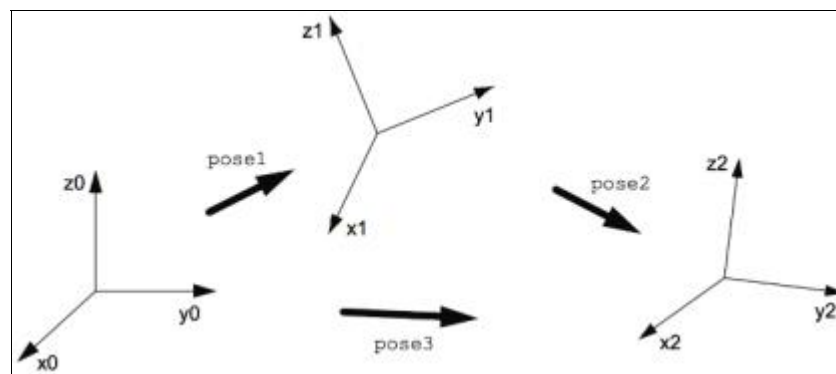
Return value

Return Value: Data type: Pose.

Example

Pose1 represents the pose of coordinate system 1 relative to coordinate system 0, and Pose2 represents the pose of coordinate system 2 relative to coordinate system 1. The pose of coordinate system 2 relative to coordinate system 0 can be calculated as follows:

pose3 := PoseMultiplication(pose1,pose2);



7.15.12. PoseInverse

Description

Used to calculate the inverse of a pose transformation.

Format

Return Value := PoseInverse(Pose1);

Parameters

Pose1: Pose 1, data type Pose.

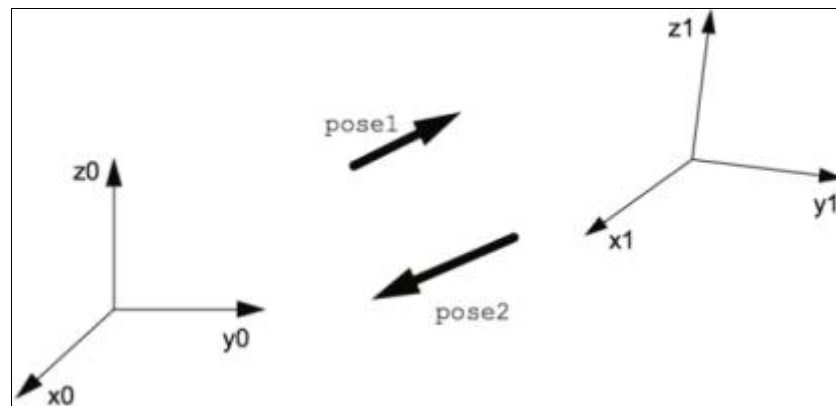
Return value

Return Value: data type Pose.

Example

Pose1 represents the pose of coordinate system 1 relative to coordinate system 0. Pose2 represents the pose of coordinate system 0 relative to coordinate system 1. If Pose1 is known, Pose2 can be calculated as:

pose2 := PoseInverse(pose1);



7.15.13. TimeStart

Description

Starts a timer.

Format


```
TimeStart(Timer Number);
```

Parameters

Timer Number: Timer ID, data type Int, range 0–7.

Example

```
TimeStart(Timer1);
```

Note

Once started, the timer value continuously accumulates and will not reset if the program stops, unless a new TimeStart command is issued.

7.15.14. TimeOut

Description

Saves the accumulated time of a timer to a specified variable.

Format

```
Return Value := TimeOut(Timer Number);
```

Parameters

Timer Number: Timer ID, data type Int, range 0–7.

Return value

Return Value: Accumulated time, data type Double, unit: seconds.

Example

```
TimerValue := TimeOut(Timer1);
```

Note

The TimeOut command can still be executed even if the timer is not started, but the value will be invalid.

7.15.15. Print

Description

Converts user-specified variables or constants to strings and outputs them to the message area for debugging and information viewing. For reference only; does not affect program logic.

Format

```
Print(var1, var2, var3, .....);
```

Parameters

var1, var2, var3, ... : Defined variables or constants (arrays are not supported). Supports Int, Float, String, and other types. The number of parameters is unlimited, but at least one parameter must be provided.

Return value

None. This function converts variables to string format and outputs them in combination for informational display.

Example

```
Print(var1, var2, var3, .....);
```

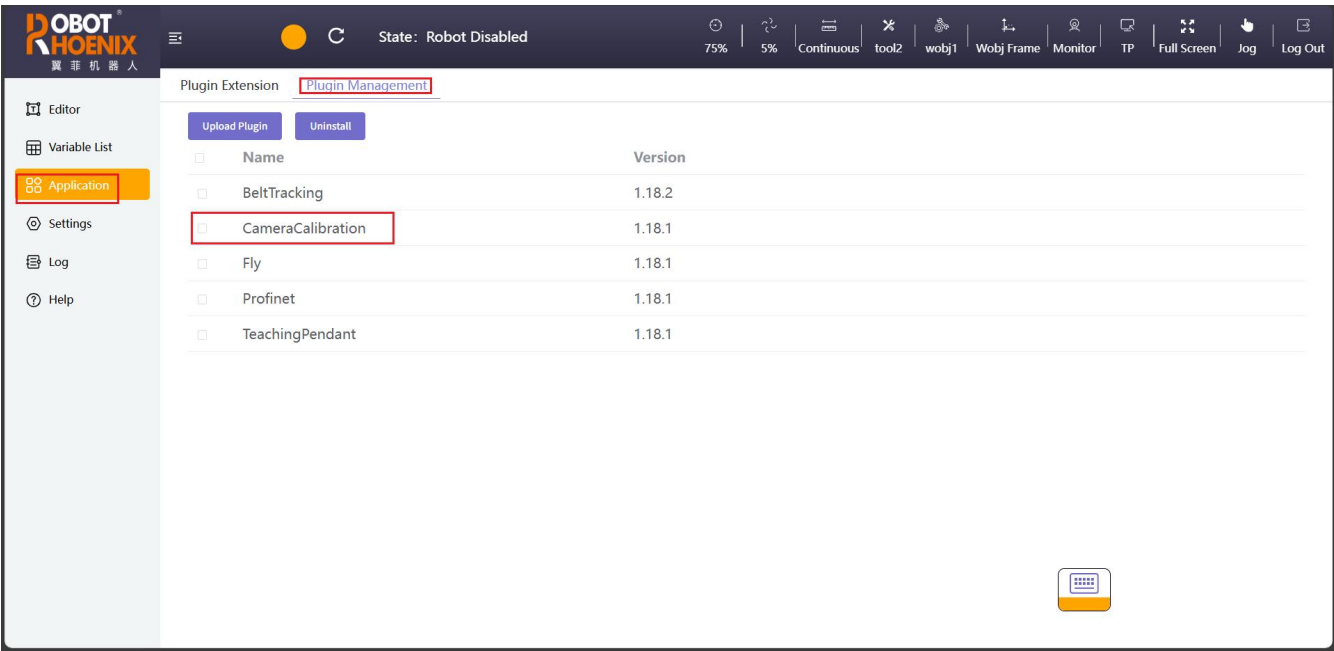
8. Plugin Extension

8.1. Vision Calibration

8.1.1. Overview

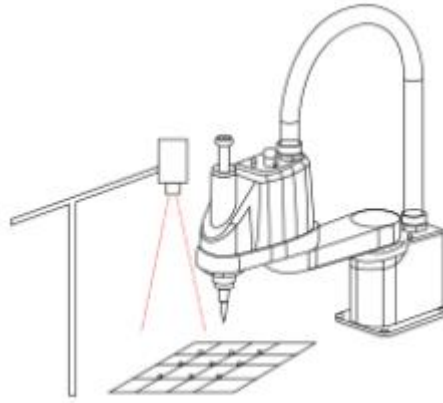
Vision calibration is a prerequisite for the robot system’s vision functionality. The calibration aims to determine the relative positional relationship between the camera and the robot, enabling the conversion of pixel coordinates to robot coordinates. This calibration supports cameras mounted externally to the robotic arm, with a fixed position relative to the robotic arm.

Before using the plugin’s extended function — Vision Calibration — you must first upload the CameraCalibration plugin via “Application” → “Plugin Management” → “Upload Plugin”, as shown in the figure below:



8.1.2. Camera Installation

Here, the SCARA robot with a fixed overhead vision calibration is used as an example. “Fixed overhead” means the camera is mounted above the calibration board in a top-down position, as shown in the figure below:



In the figure, directly below the camera is a calibration board with a 3×3 grid. Ensure that the camera, robot, and calibration board are on the same horizontal plane. The calibration board contains 9 (or 12) feature points that need to be calibrated by the robot. Make sure all feature points on the calibration board are fully visible within the camera's field of view. Install a tool on the robot's end effector, which can be a tip tool or any other tool recognizable by the vision system.

Note

Before performing vision calibration, the tool coordinate system must be calibrated first.

8.1.3. Calibration Steps

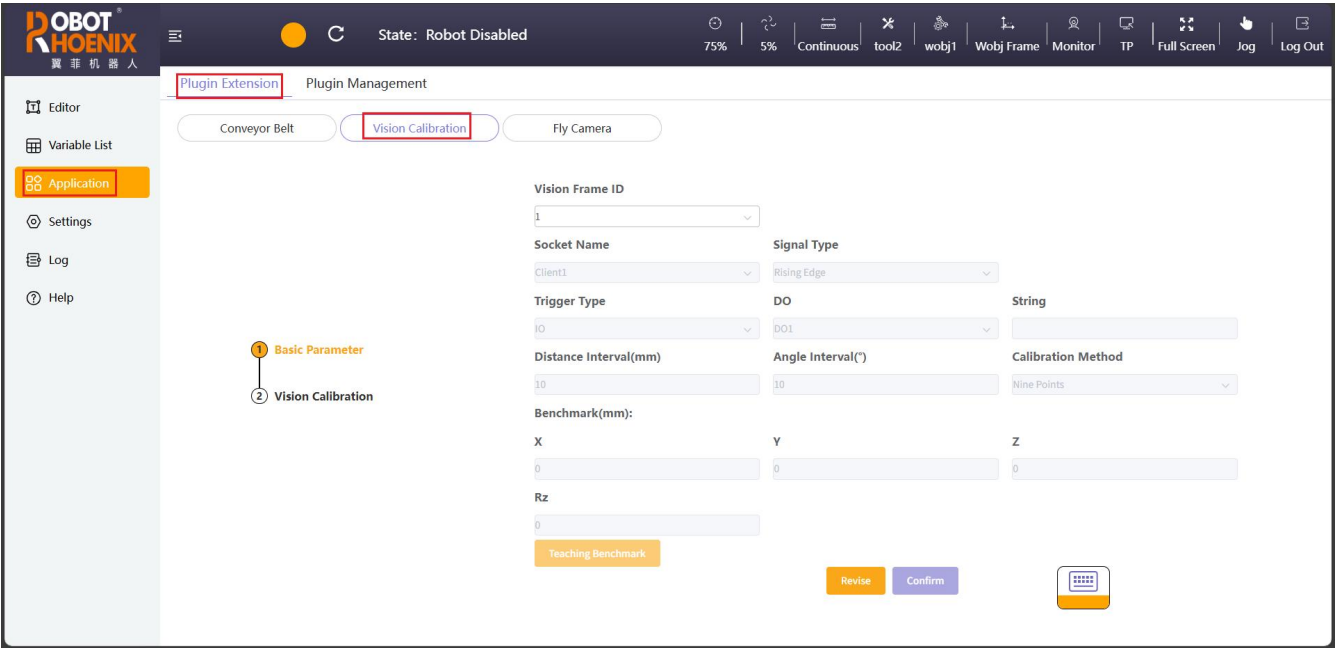
Vision calibration can be performed either automatically or manually.

For a 6-axis robot, before starting the following steps, the end effector must be calibrated or an already calibrated end effector must be used to complete the user coordinate system calibration. For a SCARA robot, the calibration can proceed directly according to the steps below.

- Manual Calibration

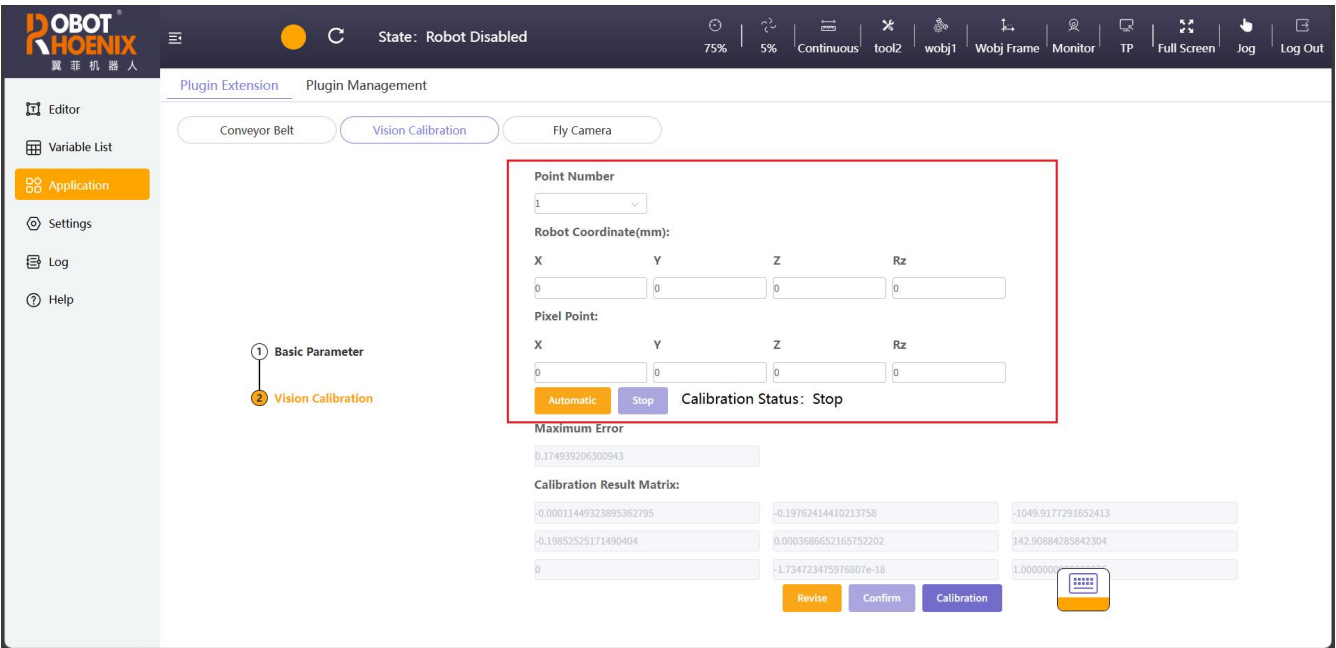
Step 1: Install the camera, adjust the robot position, place and adjust the calibration board, and configure the vision software to create the calibration template, etc.

Step 2: Select the vision coordinate system to be calibrated, as shown in the figure below.



Step 3: Choose the calibration method; both 9-point and 12-point methods are supported.

Step 4: Move the robot to each calibration point, select the corresponding point number, and manually enter the robot coordinates and pixel coordinates, as shown in the figure below.



Step 5: Click the “Calibrate” button. After calibration is complete, the maximum error and the calibration result matrix will be displayed.

Note

Click the “Modify” button to manually input or adjust the calibration result matrix. Click “OK”

to save the changes.

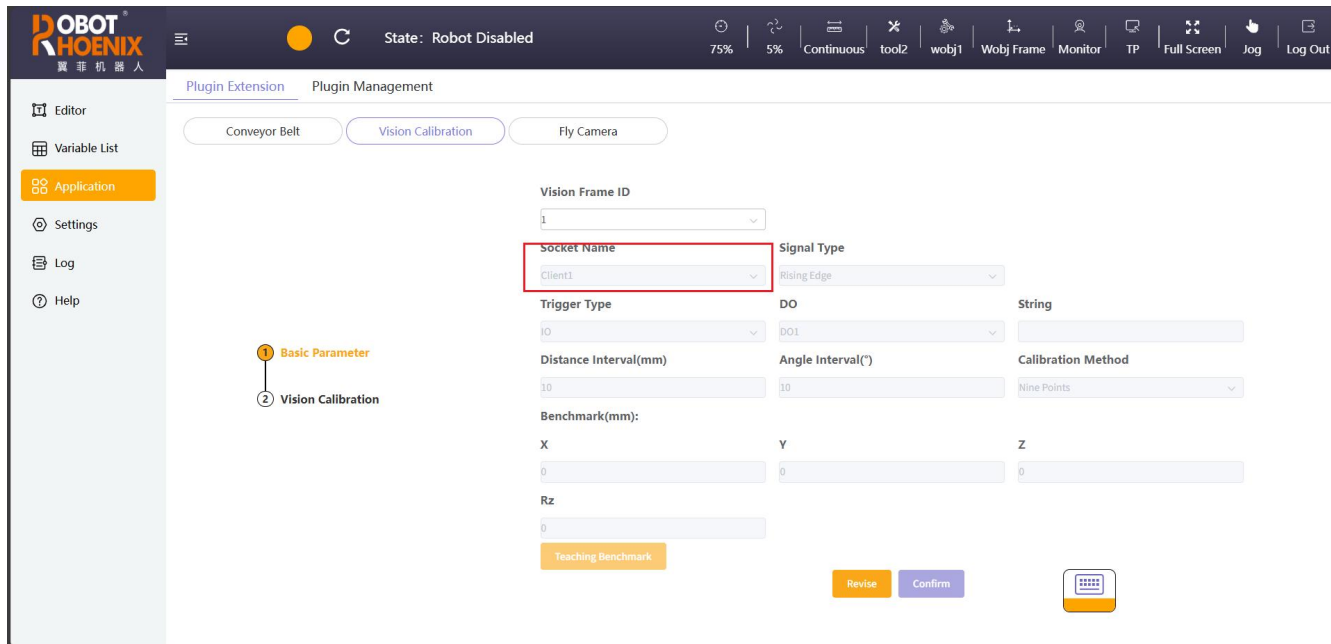
- Automatic Calibration

Step 1: Install the camera, adjust the robot position, place and adjust the calibration board, and configure the vision software to create the calibration template, etc.

Step 2: Select the vision coordinate system to be calibrated.

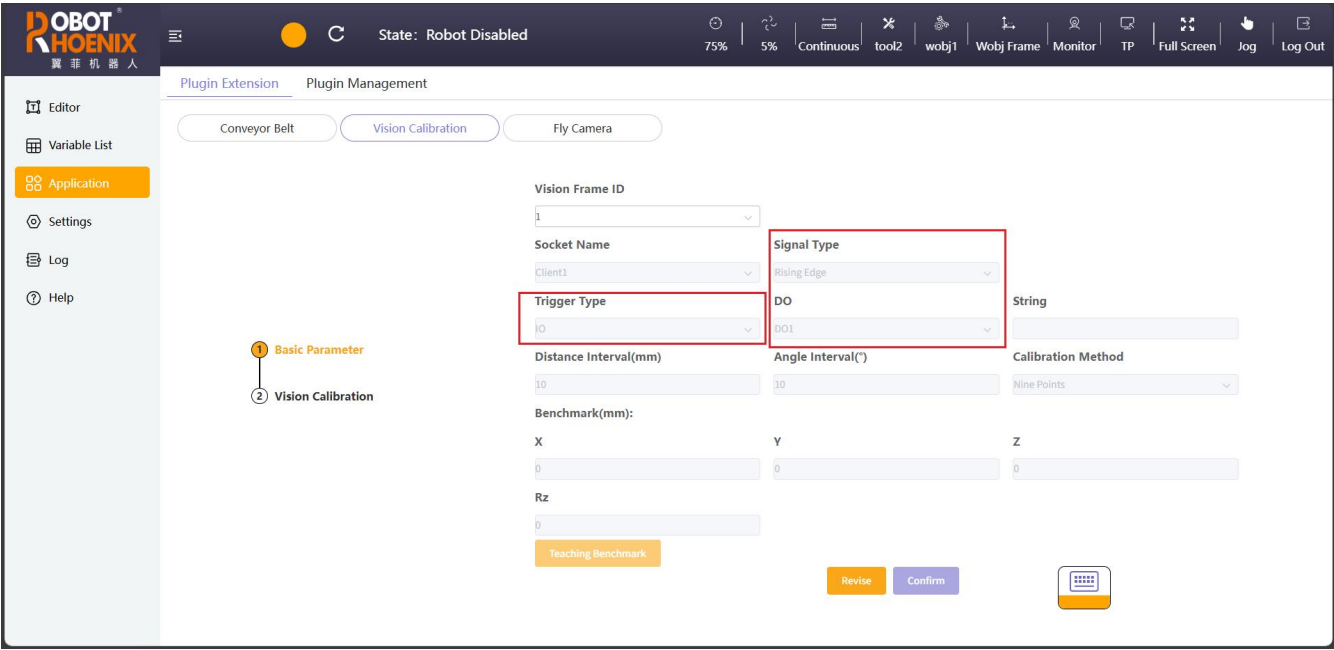
Step 3: Communication configuration. Select the Socket name that has established communication with the camera. The vision communication format is fixed:

@X1,Y1,Z1,A1,T1,X2,Y2,Z2,A2,T2.....\r. (For details, please refer to the “Commands” → “Tracking Commands” → “ConveyorVision” section.)

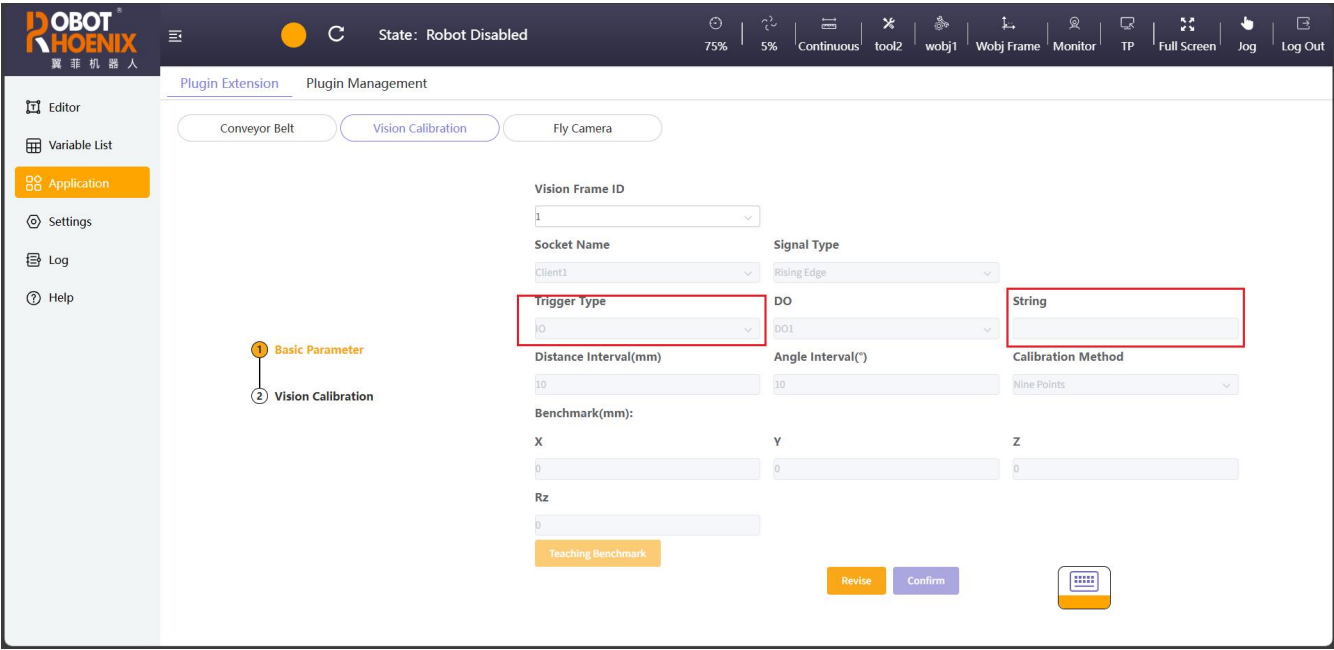


Step 4: Select the camera capture trigger method. Options include I/O trigger or Socket trigger.

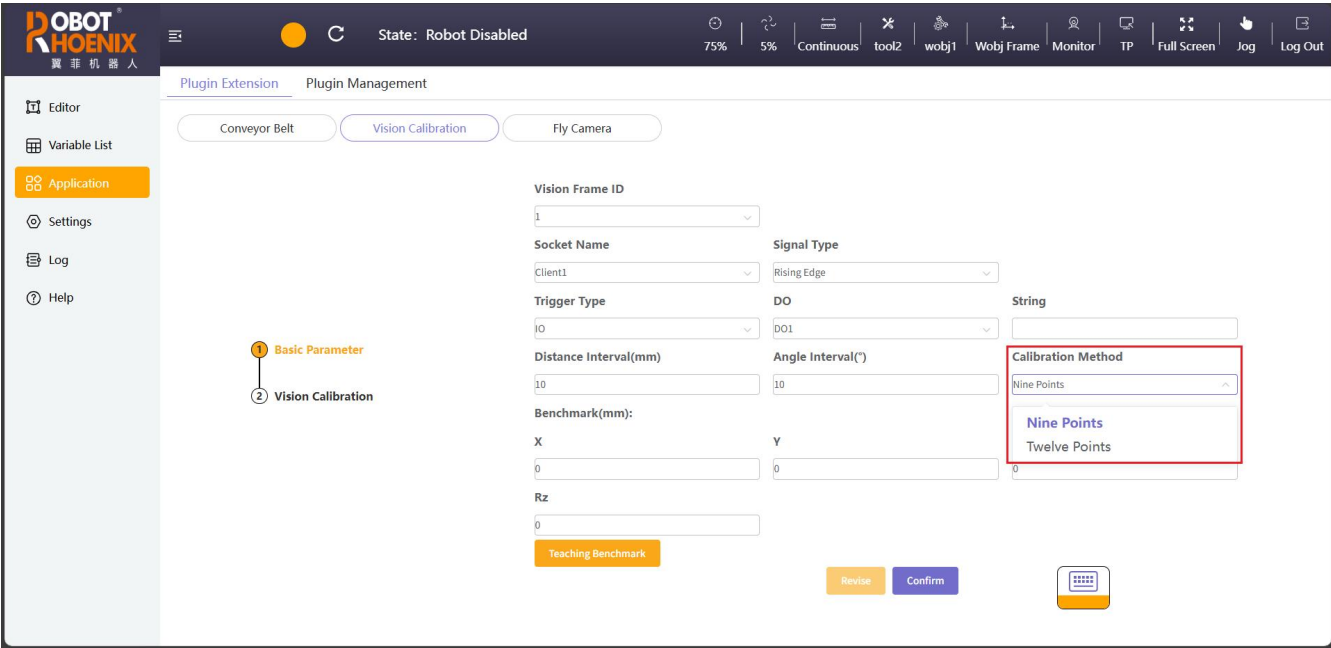
- When I/O trigger is selected, configure the required parameters as follows:



- When Socket trigger is selected, set the string that triggers the camera to capture, as shown below:



Step 5: Choose the calibration method. Options include 9-point or 12-point methods.



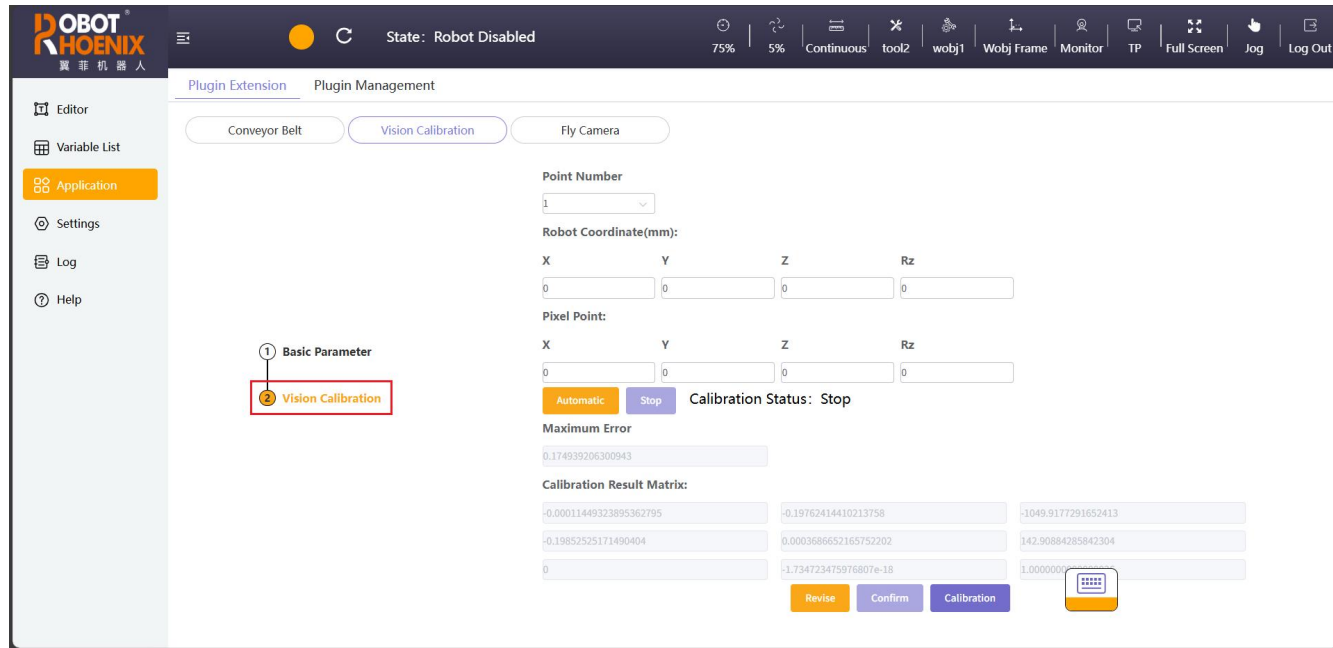
Step 6: Set the automatic calibration parameters.

Gorilla Studio robot control system supports automatic calibration. By teaching a reference point and setting the distance intervals between each point, the system automatically generates the feature point positions. The robot then moves to each feature point position and acquires the corresponding camera coordinates.

For the 9-point method, set the distance intervals between the feature points. For the 12-point method, set both the distance intervals and angle intervals between the feature points, and teach the reference point. Click OK to confirm.

Step 7: Perform automatic calibration. After calibration is complete, the maximum error and the calibration result matrix are displayed.

Click the “Auto” button to switch the calibration status to “Calibrating”. The robot automatically moves to each calibration point and acquires robot and pixel coordinates. To stop the calibration, click the “Stop” button; the status will switch to “Stopped”.

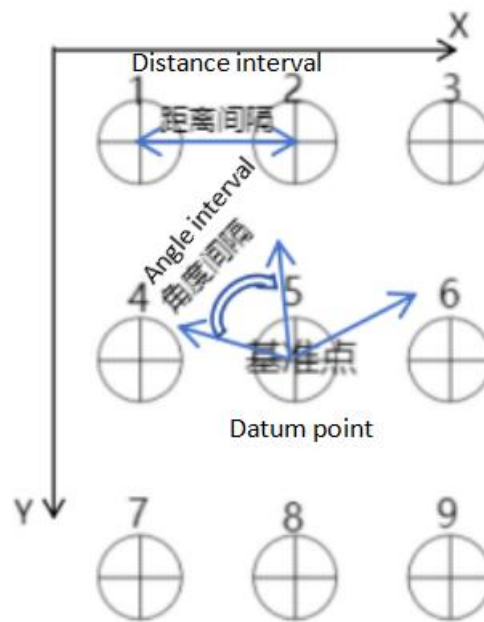


Note:

Click the “Modify” button to manually input or adjust the calibration result matrix, and click “OK” to save.

If the Stop button is clicked during automatic calibration, clicking “Auto” again will restart calibration from the first calibration point.

The sequence of calibration points, datum point, distance intervals, and angle intervals are defined as shown in the figure below:

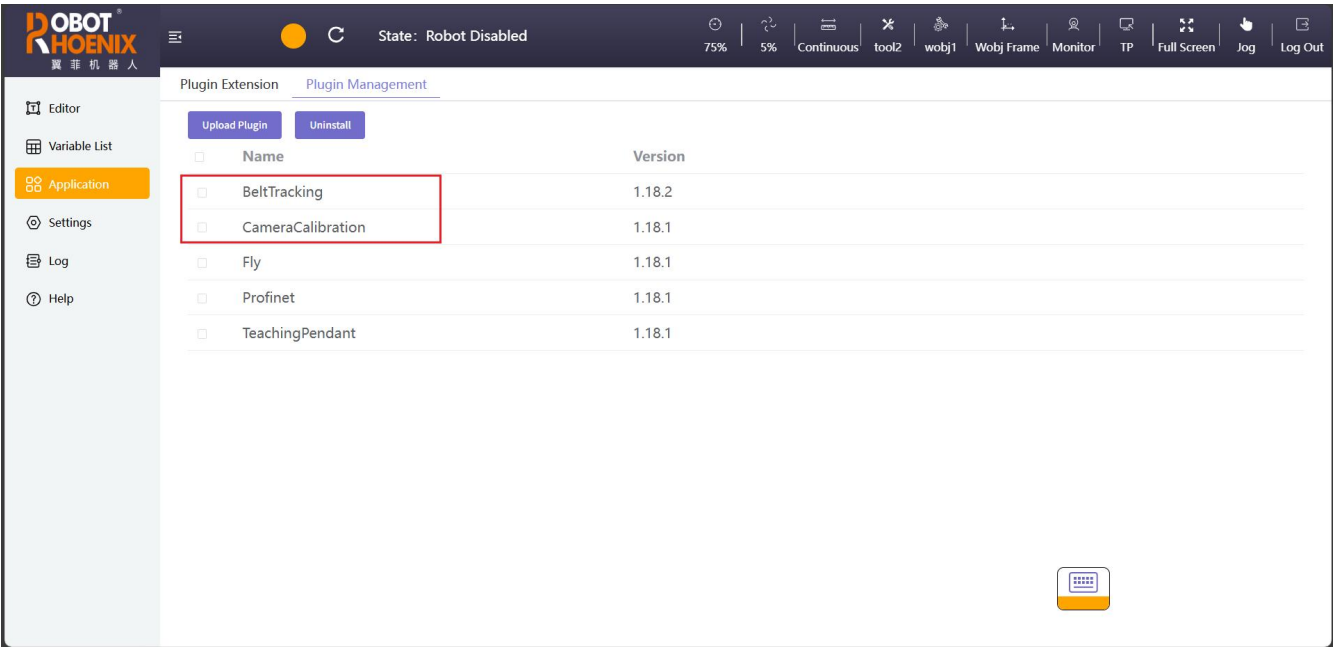


8.2. Conveyor

8.2.1. Overview

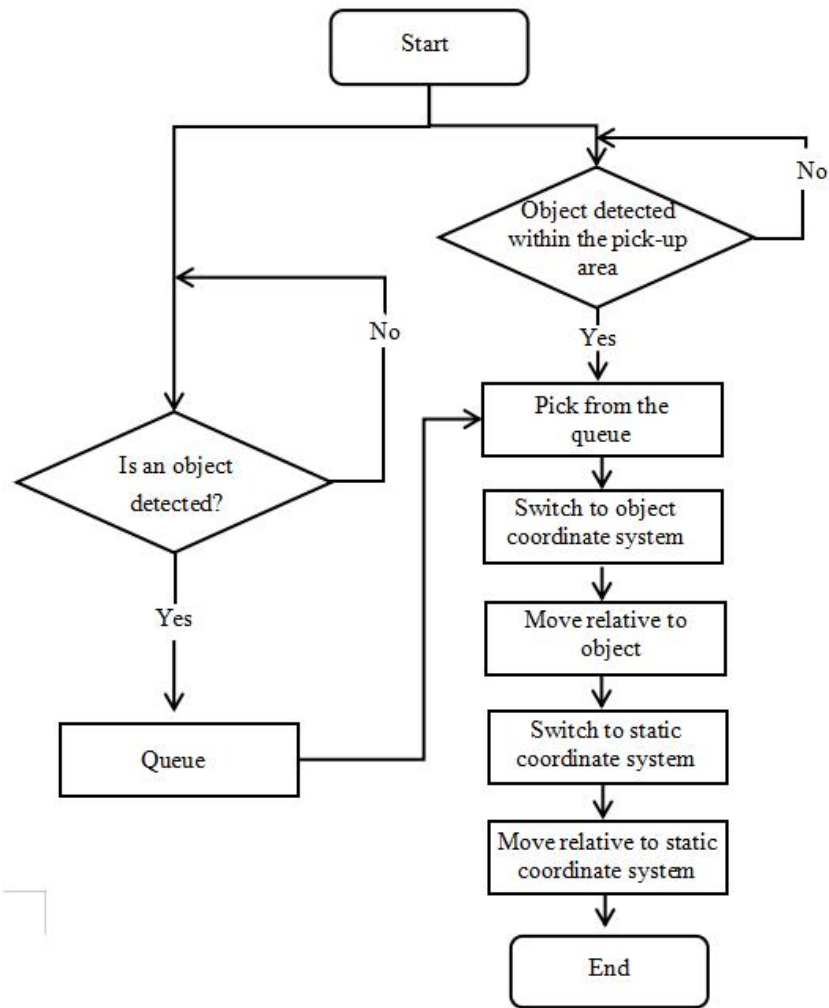
Conveyor Tracking detects objects on a conveyor belt using vision or photoelectric sensors, allowing the robot to accurately grasp them while they are in motion. This is typically used when the robot needs to pick or place workpieces on a moving conveyor or perform operations such as spraying following the workpieces on the conveyor.

Before using the plugin’s extended function — Conveyor, you must first upload the “BeltTracking” and “CameraCalibration” plugins via “Application” → “Plugin Management” → “Upload Plugin”, as shown in the figure below:



8.2.1.1. Workflow

The conveyor tracking process can be divided into two parts: detection and motion tracking. Detection obtains the position of objects on the conveyor using vision or photoelectric sensors, inputs the acquired object position information into the conveyor object queue, and tracks the position changes through encoder feedback. Motion tracking is the robot movement using the moving objects in the queue as the reference coordinate system. Detection and motion tracking are processed in parallel. The specific workflow is shown in the figure below.

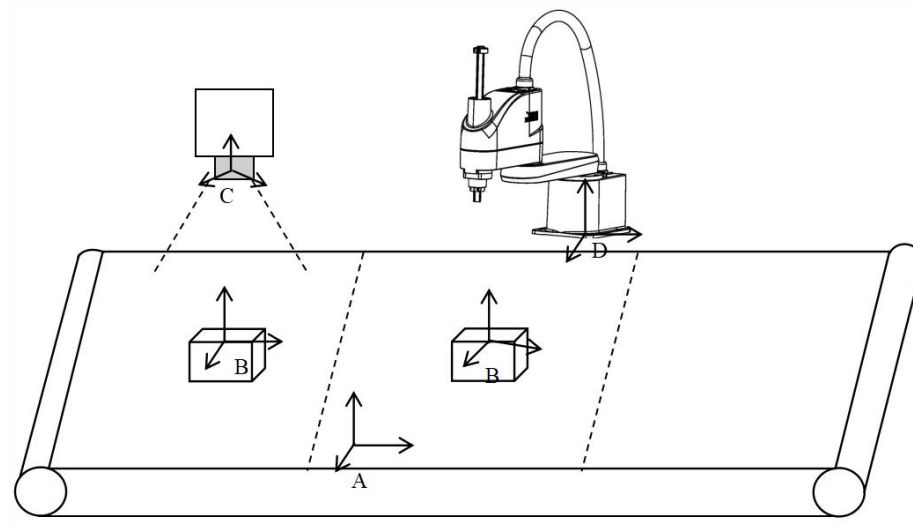


8.2.1.2. Description of Tracking Coordinate System

Tracking motion is performed with the object coordinate system as the reference frame. The coordinate systems involved in the tracking process are described as follows:

- Conveyor Coordinate System (A): Used to describe the position and orientation of the conveyor relative to the robot base coordinate system (D). It is a static coordinate system, and the positive direction of its X-axis aligns with the conveyor's moving direction.
- Object Coordinate System (B): Used to describe the position and orientation of the object on the conveyor. It represents the object's reference point expressed in the conveyor coordinate system (A). This coordinate system is rigidly attached to the object, moves together with it, and is therefore a dynamic coordinate system.

- Vision Coordinate System (C): Used to describe the positional transformation relationship between the camera and the conveyor. It obtains the instantaneous position of the object on the conveyor through the camera, and acquires the object's movement on the conveyor in real time through the encoder, thereby enabling the robot to perform real-time tracking of the object's position.



8.2.1.3. Functional Specifications

- Conveyor types: linear type, disc type.
- Number of conveyors: configurable up to 4.
- Detection method: vision or photoelectric sensor.
- Tracking motion commands: supported commands include MoveL, MoveC, and MoveLJump.
- Vision data type: robot coordinates or pixel coordinates. If robot coordinates are selected, coordinate conversion is required. Based on the calibration relationship of the vision coordinate system, robot coordinates are converted into pixel coordinates. If pixel coordinates are selected, no coordinate conversion is needed.
- Vision communication format:

When objects are detected: @X1,Y1,Z1,A1,T1,X2,Y2,Z2,A2,T2.....\r

When no object is detected: NG (For details, refer to “Commands” → “Tracking Commands”)

→ “ConveyorVision”.)

- Number of objects per image: 0 to 10.
- Object queue capacity: 500 objects.
- Number of robots assigned per conveyor: 1.

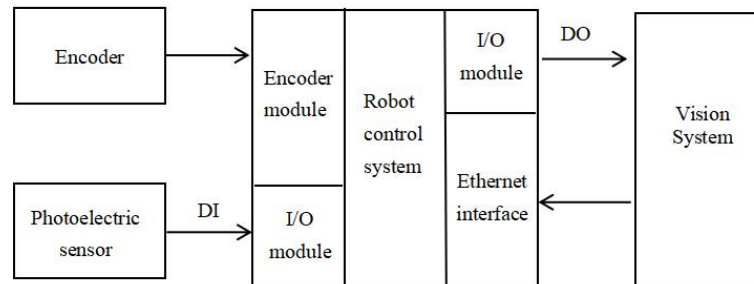
8.2.1.4. Configuration Procedure

When using the conveyor tracking process, the following tasks are required. For detailed information on each part, please refer to the corresponding sections below:

- (1) Hardware physical connections
- (2) Coordinate system settings
- (3) Conveyor parameter settings
- (4) User program writing

8.2.2. Hardware Configuration

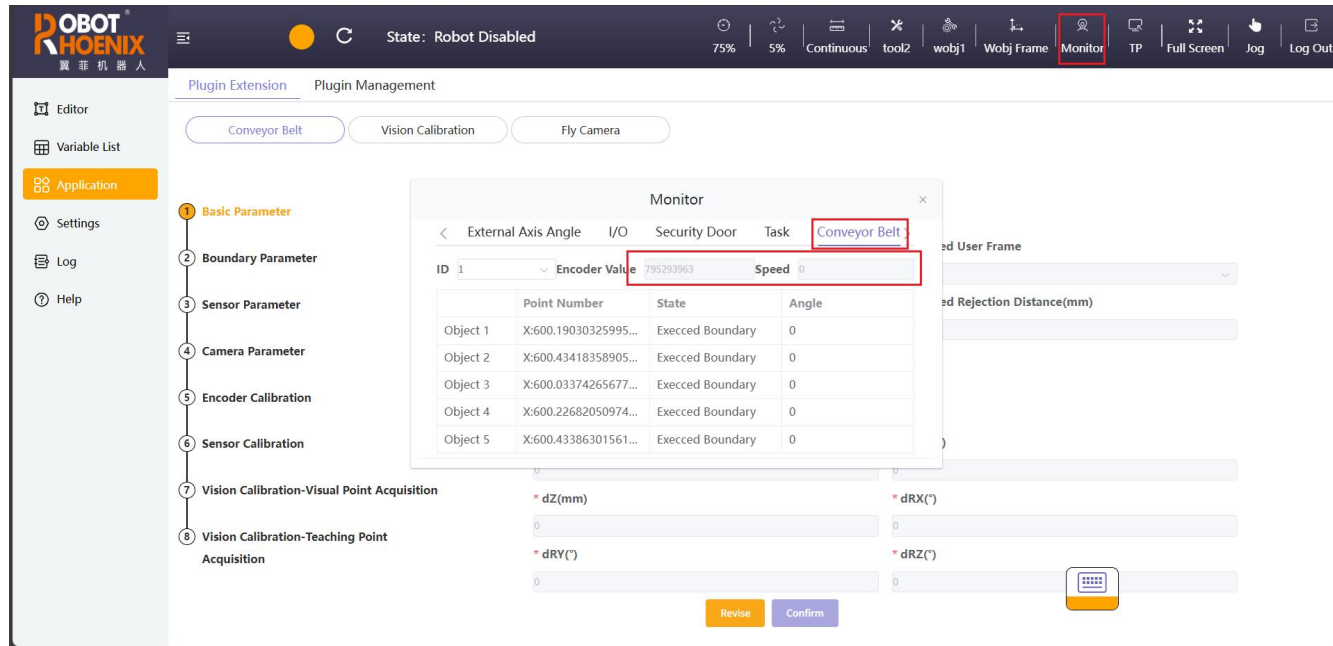
The hardware configuration for the conveyor tracking process is shown in the figure below. The photoelectric sensor and the vision system are optional, depending on the selected detection method. If vision detection is used, the camera must be triggered via a hardware trigger.



8.2.2.1. Encoder

A pulse-type encoder must be used. Connect the encoder signal wires and power wires to the encoder interface inside the control cabinet. After completing the physical wiring, rotate the

encoder and check the encoder value change under “Monitoring” → “Conveyor” to verify whether the physical wiring is correct, as shown below:



8.2.2.2. Photoelectric Sensor

The photoelectric sensor outputs different voltage levels when triggered by an object. The robot controller detects objects via signal edge changes. Connect the sensor signal wire to a standard DI port on the I/O module inside the control cabinet. After wiring, verify proper operation by checking whether the corresponding DI signal changes in the monitoring interface.

8.2.2.3. Vision

Interaction between the vision system and the robot includes two parts: DO triggering and Ethernet communication. For DO triggering, a signal cable needs to be connected from the DO interface on the robot control cabinet to the camera's IO trigger interface. After the physical connection is completed, you should manually output the DO signal and check whether the camera can be triggered normally.

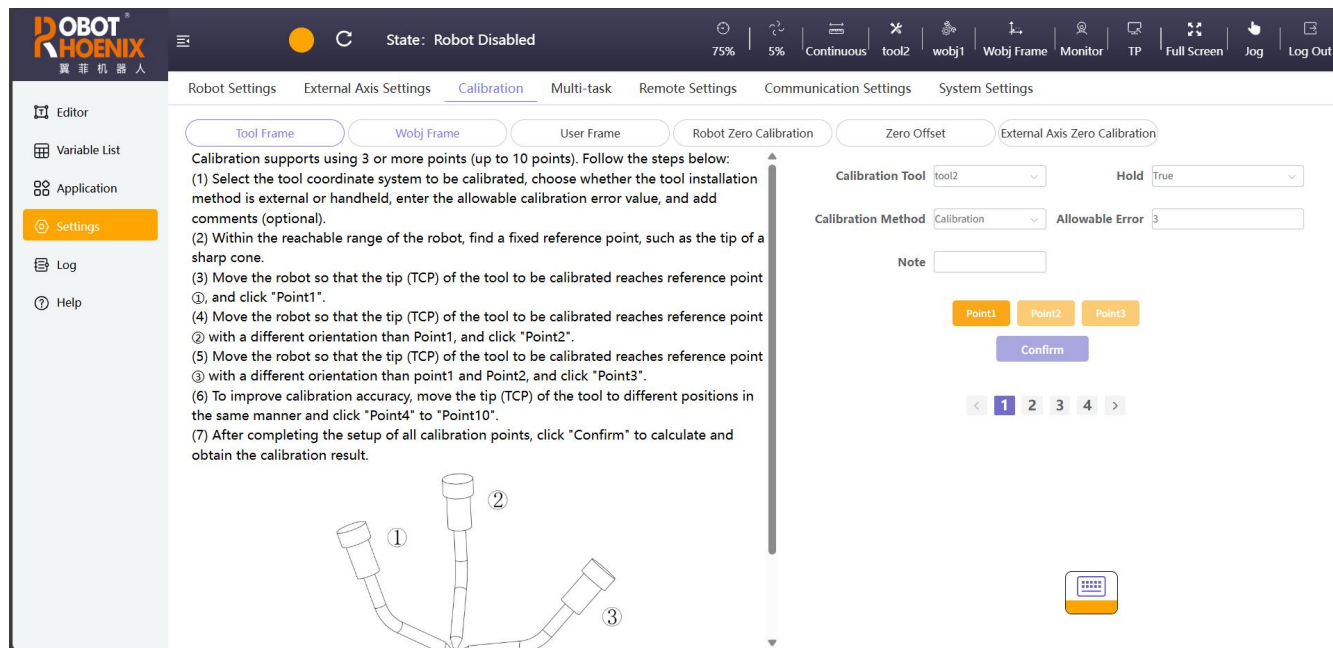
8.2.3. Coordinate System Settings

8.2.3.1. Tool Coordinate System Calibration

After installing a tool such as a gripper, suction nozzle, or welding torch on the robot's end-effector, tool coordinate system calibration must be performed before use. This calibration determines the position and orientation of the tool relative to the robot's base coordinate system, so that the position and orientation of the robot tool end effector can be precisely controlled through programming to perform the operation.

Operation method

As shown in the figure below, click “Settings” → “Calibration” → “Tool Coordinate System” in the left navigation bar to enter the tool coordinate system calibration interface, and select the tool to be calibrated.



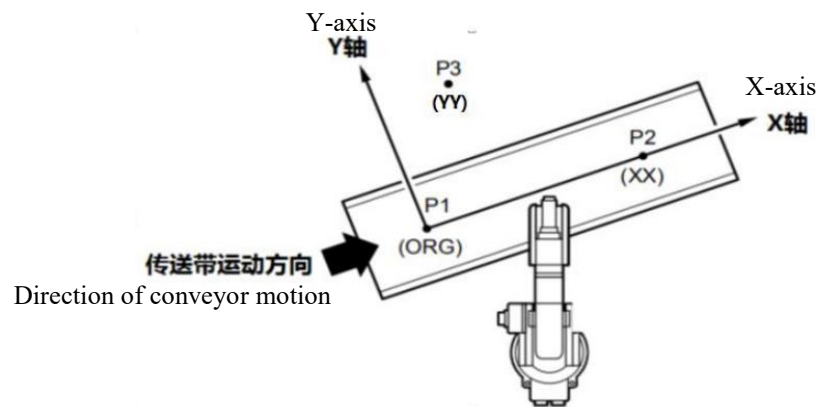
The tool coordinate system IDs range from 1 to 16. ID 1 is reserved for the case when no tool is installed and is also the default tool ID. The other 15 coordinate systems can be assigned to actual tools and calibrated accordingly.

Tool calibration methods include manual input, 3-point method, 4-point method, 6-point method, and calibration.

For 6-axis robots, the 3-point, 4-point, or 6-point methods are used to calibrate the position and orientation of the tool coordinate system. For 4-axis robots, the calibration method is used to calibrate the X and Y coordinates of the tool coordinate system. For DIAMOND models, the calibration method is used to calibrate the X and Y or X and Z coordinates of the tool coordinate system. For detailed steps, please refer to the “Calibration → Tool Coordinate System Calibration” section.

8.2.3.2. Conveyor User Coordinate System Calibration

The conveyor coordinate system is a special-purpose user coordinate system. Its parameters must be set or calibrated via “Settings” → “Calibration” → “User Coordinate System Calibration”. The conveyor coordinate system defines the position and orientation of the conveyor, with the positive X-axis aligned with the conveyor’s movement direction, as shown in the figure below.



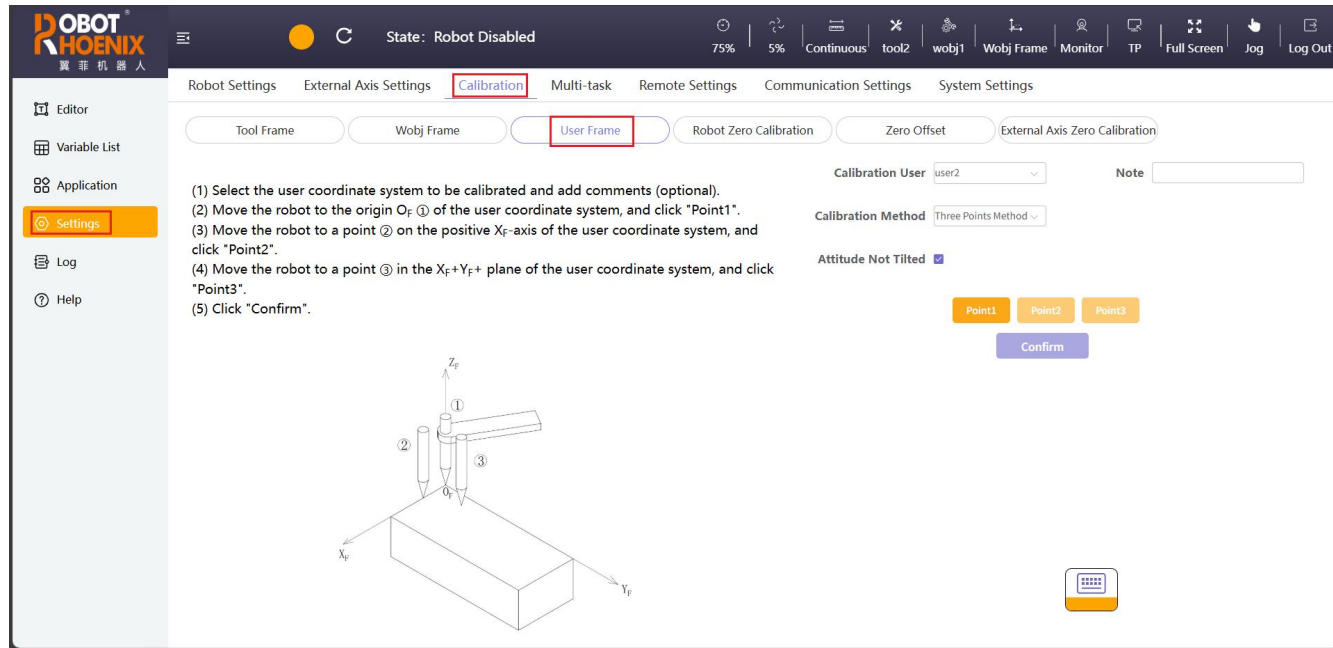
Note

When calibrating a user coordinate system, it is recommended to use a standard calibration cone to avoid affecting tracking accuracy.

The X-axis of the user coordinate system of the conveyor must be aligned with the conveyor’s positive movement direction; otherwise, the robot will not be able to correctly execute tracking motions.

Operation method

As shown in the figure below, click “Settings” → “Calibration” → “User Coordinate System” in the left navigation bar to enter the user coordinate system calibration interface and perform user coordinate system calibration specified by the conveyor belt process.

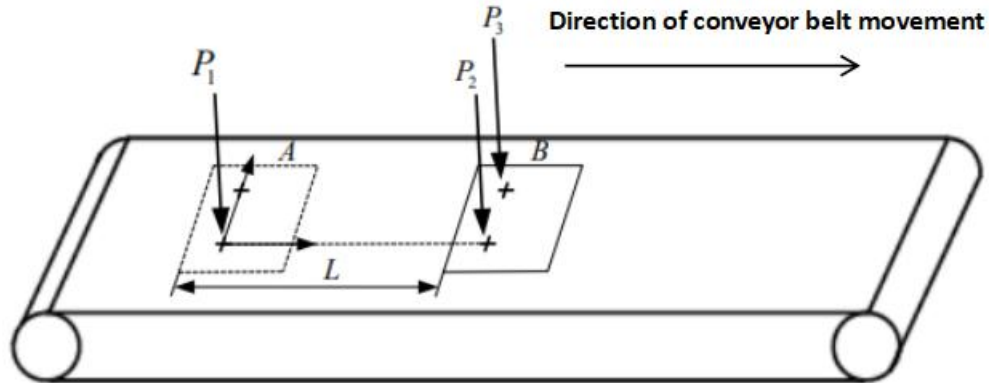


Gorilla Studio robot control system supports both linear conveyors and disc-type conveyors, each requiring a different calibration method.

8.2.3.2.1. Linear Conveyor

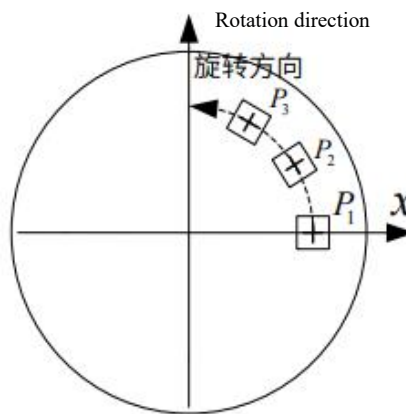
Calibration is performed using the 3-point method. The three points are selected as shown in the figure below: Place the calibration board or another calibration reference on the conveyor, align the robot end-effector with the reference point, and click “Get Point 1”; Point 1 will serve as the origin of the conveyor coordinate system. Keep the calibration board fixed relative to the conveyor while moving the conveyor a distance L in the positive direction so that P1 moves to P2. The vector P1–P2 defines the positive X-axis; then click “Get Point 2”. Keep the conveyor stationary, with the Z-axis perpendicular to the conveyor surface and pointing upward. According to the right-hand rule, select a point P3 within the X+Y+ quadrant and click “Get Point 3”. Click “OK” to complete the calibration.

To verify whether the conveyor user coordinate system is correct, change the jogging coordinate system to the user coordinate system and set the conveyor user coordinate system as the currently used user coordinate system. Jog the robot along the X, Y, and Z directions, and check whether the X direction is the positive direction of conveyor movement and whether the Z direction is perpendicular to the conveyor plane and pointing upward.



8.2.3.2.2. Disc-Type Conveyor

Calibration is performed using the rotation method. The selection of three points (or more, up to 30 points; for details, refer to the rotation method) is shown in the figure below: Fix the calibration reference point on the disc. Rotate the reference point to an appropriate position and pick the first point P_1 by clicking “Get Point 1”. The direction from the rotation center to P_1 defines the positive X-axis of the coordinate system. Rotate the disc to pick points P_2 and P_3 sequentially, then click “OK”. Three points define an arc, and the center of the arc serves as the origin of the coordinate system.



Note

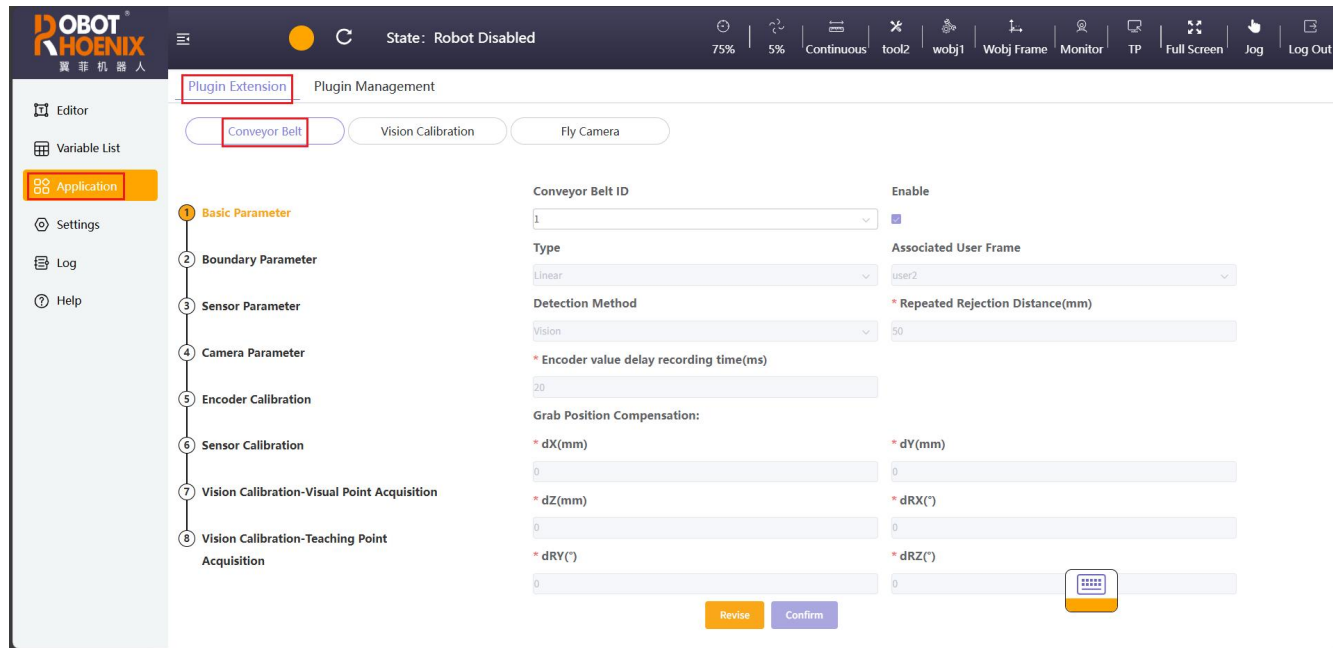
To improve accuracy, it is recommended that the three points be spaced as far apart as possible within the X+Y+ quadrant.

8.2.4. Parameter Settings

8.2.4.1. Basic Parameter Settings

Operation Method

Click “Application” → “Plugin Extensions” → “Conveyor” → “Basic Parameters” in the left navigation bar to enter the conveyor basic parameter settings interface, as shown in the figure below:



- Conveyor ID

Click the drop-down menu for “Conveyor ID” and select the ID. Allowed values range from 1 to 4.

- Enabled

Check the box to enable the conveyor; uncheck to disable it.

- Type

Click the drop-down menu for “Type” and select either “Linear” or “Disc”.

- Associated User Coordinate System

Click the drop-down menu for “Associated User Coordinate System” and select the calibrated user coordinate system ID. Allowed values range from user1 to user16.

- Detection Method

Click the drop-down menu for “Detection Method” and select either “Vision” or “Sensor”.

- Duplicate Removal Distance

To prevent the same object from being detected repeatedly and pushed into the object queue, if the newly detected object position is closer than the duplicate removal distance to an existing object in the queue, it will be considered a duplicate and will not be added to the queue.

Enter the duplicate removal distance in the input box. Allowed values range from 0 to 9999 mm.

- Pick Position Compensation

Set a uniform offset for all target positions in tracking motion. This offset is defined relative to the work object coordinate system.

dX, dY, dZ: Data type Double, allowed range -1999 to 9999, unit: mm

dRx, dRy, dRz: Data type Double, allowed range -180 to 180, unit: °

8.2.4.2. Boundary Parameter Settings

Operation method

Click “Application” → “Plugin Extensions” → “Conveyor” → “Boundary Parameters” in the left navigation bar to enter the conveyor boundary parameter settings interface.

- Working Upper Limit

The highest reachable boundary for the robot, which also represents the upper boundary of the object search area. Objects that pass this boundary can be detected by the conveyor query command. The upper work limit is represented by the X coordinate in the conveyor coordinate system. Ensure that the upper work limit is within the robot’s workspace to avoid singular positions.

Data type: Double. Linear conveyor allowed range: -100000 to 100000, unit: mm. Disc-type conveyor allowed range: [0, 360), unit: °.

- Pick Lower Limit

The lower boundary of the object-picking area on the conveyor. If an object moves beyond this boundary before being picked, it will no longer be detected and will be removed from the

object queue.

Data type: Double. For linear conveyors, the valid range is -100000 to 100000 , unit: mm. For rotary (disc-type) conveyors, the valid range is 0 to 360 , unit: $^{\circ}$.

- Working Lower Limit

The lowest reachable boundary for the robot.

Data type: Double. For linear conveyors, the valid range is -100000 to 100000 , unit: mm. For rotary conveyors, the valid range is 0 to 360 , unit: $^{\circ}$.

Note:

If an object moves beyond the working lower limit during tracking or synchronous motion, the robot system will generate an error and stop. To reduce the chances of this occurring, increase the distance between the pick lower boundary and the working lower limit to ensure the object does not cross the working lower limit from the start of the pick to the successful grasp.

- Virtual Boundary

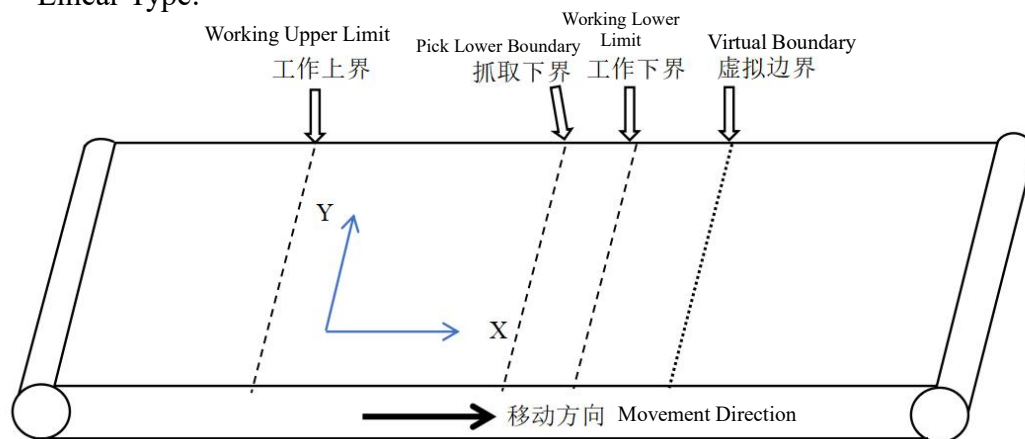
The cutoff boundary for updating object information. Once an object crosses this boundary, it will be removed from the object queue.

Data type: Double. For linear conveyors, the valid range is -100000 to 100000 , unit: mm. For rotary (disc-type) conveyors, the valid range is 0 to 360 , unit: $^{\circ}$.

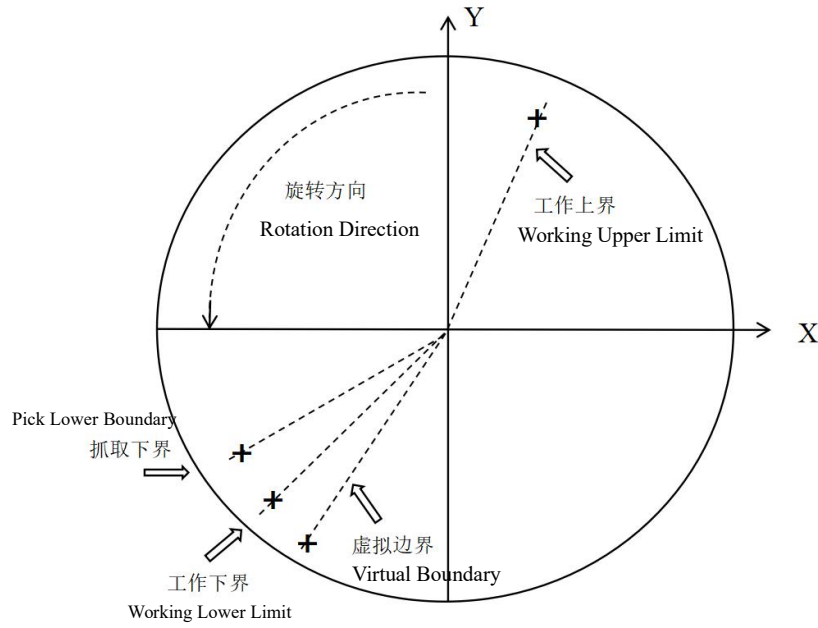
Description

The working upper limit, pick lower boundary, working lower limit, and virtual boundary are four lines perpendicular to the X-axis of the conveyor coordinate system and are defined by their X-coordinates.

- Linear Type:



- Rotary (Disc) Type:



Note

The boundary parameters must satisfy the following relationship:

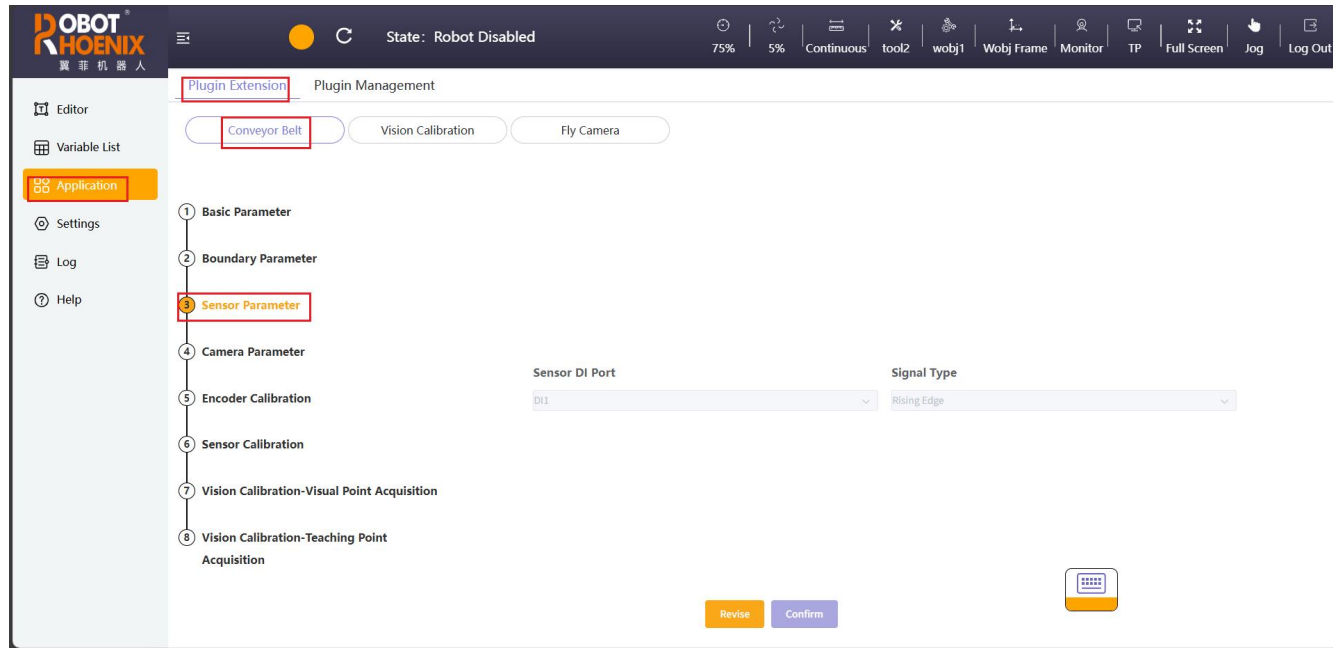
Working Upper Limit < Pick Lower Boundary < Working Lower Limit < Virtual Boundary.

8.2.4.3. Sensor Parameter Settings

This setting is required when the detection method is set to “Sensor.”

Operation method

Click on the left navigation panel: Application → Plugin Extension → Conveyor → Sensor Parameters to enter the sensor parameter settings interface.



- **Sensor DI Port**

The input port number of the photoelectric sensor on the I/O module.

Valid range: DI1 to DI16.

- **Signal Type**

The signal edge output by the sensor when triggered by a workpiece.

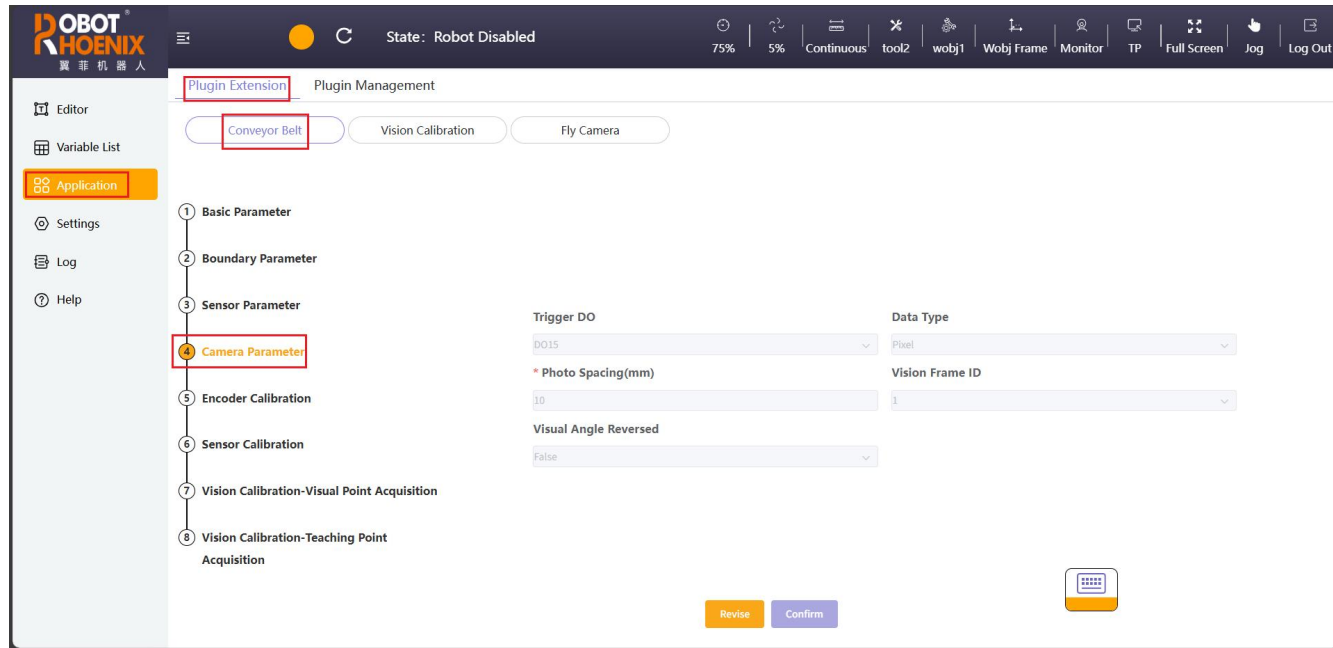
Options: Rising Edge or Falling Edge.

8.2.4.4. Camera Parameter Settings

This setting is required when the detection method is set to “Vision.”

Operation method

Click on the left navigation panel: Application → Plugin Extension → Conveyor → Camera Parameters to enter the camera parameter settings interface.



- Camera Trigger DO

The input signal on the I/O module used for hardware triggering the camera.

Valid range: DO1 to DO16.

- Camera Data Type

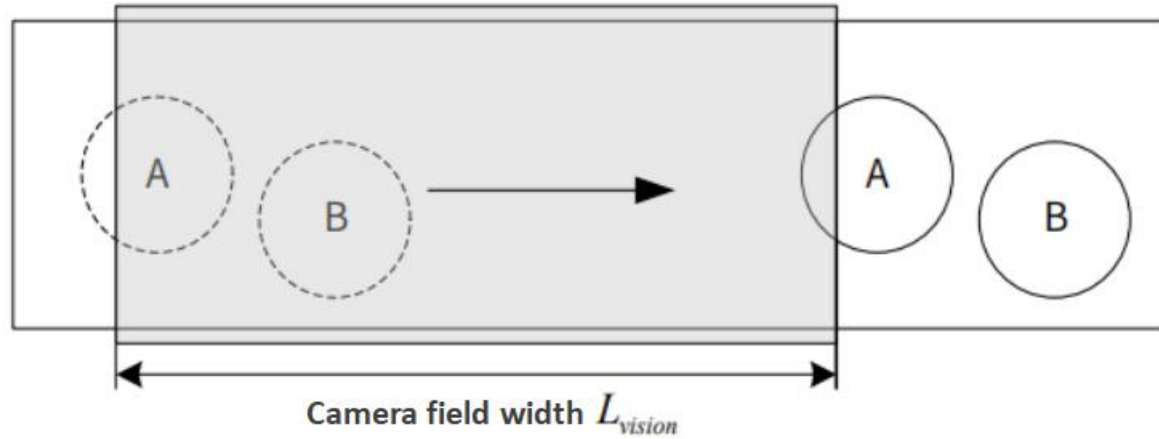
Select “Pixel” if the data sent to the robot is in pixels; otherwise, select “Robot Coordinate”.

- Photo Interval

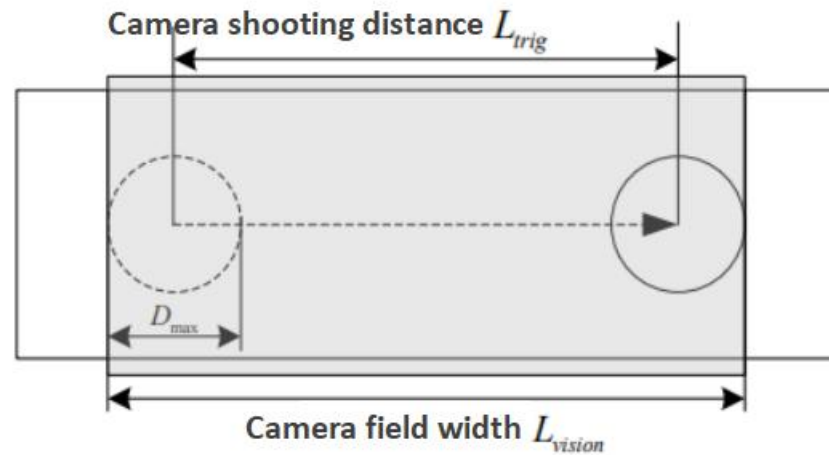
The conveyor vision system triggers a photo based on the conveyor’s movement distance. For every movement equal to the “Photo Interval,” the controller sends a DO signal to the camera to trigger shooting.

Data type: Double. Linear conveyor: valid range 0 to 1999, unit: mm. Rotary (disc) conveyor: valid range 0 to 360, unit: °.

The principle for setting the photo interval is to ensure that every object on the conveyor is fully captured without missing any. If the photo interval is set equal to the camera’s field of view, the images captured by the camera in two consecutive shots will seamlessly connect, capturing every part of the conveyor belt. However, if an object happens to be located at the junction between two consecutive shots, each of the two shots will capture only half of the object, making it unrecognizable. As shown in the figure, during the first shot, the workpiece is at the dashed position; during the second shot, the workpiece has moved to the solid line position on the right. In this case, neither of the two consecutive shots can recognize object A.



To avoid this situation, the two consecutive images should slightly overlap, with the overlap width greater than the maximum object width, ensuring that each object is fully captured in at least one image, as shown in the figure below.



The vision system must complete image recognition within one photo cycle, where Photo Cycle = Photo Interval / Conveyor Speed. A smaller photo interval shortens the photo cycle and requires faster vision processing. Therefore, set the photo interval as large as possible without missing objects. Considering all requirements, the photo interval should not be too large or too small. It can be calculated as: L_{trig} = Photo Interval, L_{vision} = Field of View Width, D_{max} = Maximum Width of the Object Recognition Mark, δ_{vision} = Safety Margin

$$\begin{cases} L_{trig} = L_{vision} - D_{max} - \delta_{vision} \\ \delta_{vision} < 10mm \end{cases}$$

- Vision Coordinate System ID

If the data sent to the robot is in pixels, vision calibration and coordinate transformation must

be performed in the controller (via “Plugin Extension” → “Conveyor” → “Vision Calibration” → “Vision Point Acquisition”). The calibration results are stored in the vision coordinate system list, and this parameter specifies the ID of the vision coordinate system.

Valid range: 1 to 16.

- Invert Vision Angle

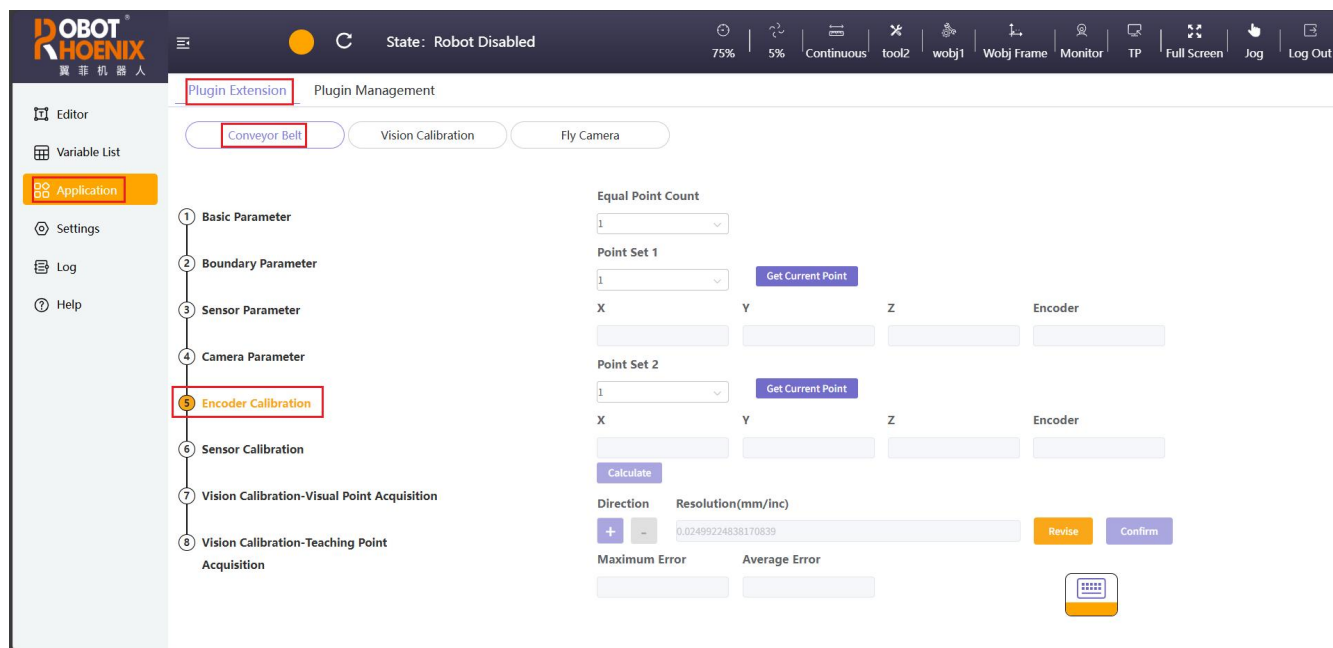
Applicable when the vision recognition direction is opposite to the robot coordinate system.

Enabling this option automatically inverts the received angle values before operation to ensure correct grasping posture.

Options: False = Disabled; True = Enabled.

8.2.4.5. Encoder Calibration

Complete the encoder resolution and direction settings in the encoder calibration interface.



Calibration must be performed in the base coordinate system, ensuring that the robot’s end calibration cone is aligned with the centerline of the flange. If it is not on the flange centerline, tool coordinate system calibration must be performed first, and this tool coordinate system should be selected when jogging the robot.

Encoder Resolution: The movement increment of the conveyor corresponding to one encoder pulse (in mm or °).

Encoder Direction: Indicates whether the encoder pulse value increases or decreases when the conveyor moves in the positive direction.

Operation method

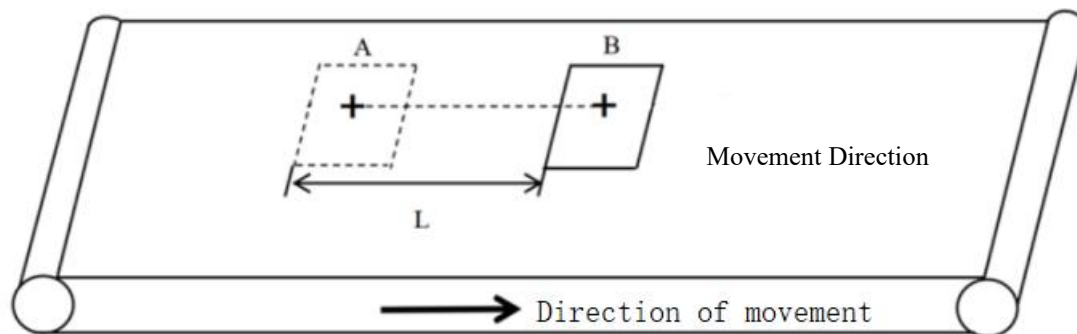
● Linear Conveyor (Supports Multi-Point Calibration)

The conveyor is divided into two calibration zones: Zone A and Zone B. Each zone supports collecting 1 to 9 points for calibration. Perform the steps below:

Step 1: In calibration Zone A, sequentially collect 1–9 points. Align the tip of the robot's end calibration cone with each calibration point, select the corresponding number 1–9 in "Point Set 1," and click "Get Current Point".

Step 2: Move the conveyor to calibration Zone B and similarly collect 1–9 points. Align the robot's end calibration cone with each calibration point, select the corresponding number 1–9 in "Point Set 2," and click "Get Current Point".

Step 3: After collecting points in both zones, click "Calculate". The system will automatically compute the encoder resolution and direction, completing the calibration.



● Rotary (Disc) Conveyor (Supports Multi-Point Calibration)

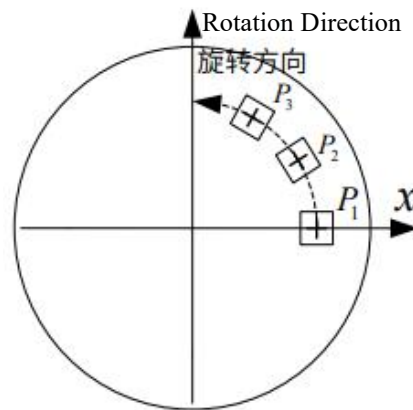
The disc conveyor supports collecting 1 to 9 points for calibration. Perform the steps below:

Step 1: Place a fixed reference point on the disc conveyor. Align the tip of the robot's end calibration cone with this point, select number 1, and click "Get Current Point".

Step 2: Keep the reference point fixed relative to the conveyor and rotate the conveyor. Each time the reference point reaches a new angular position, align the robot's end calibration cone with the point, sequentially select numbers 2–9, and click "Get Current Point" to record.

Step 3: After all points are collected, click "Calculate". The system will automatically compute the encoder resolution and direction based on the collected multi-point data,

completing the calibration.



8.2.4.6. Sensor Calibration

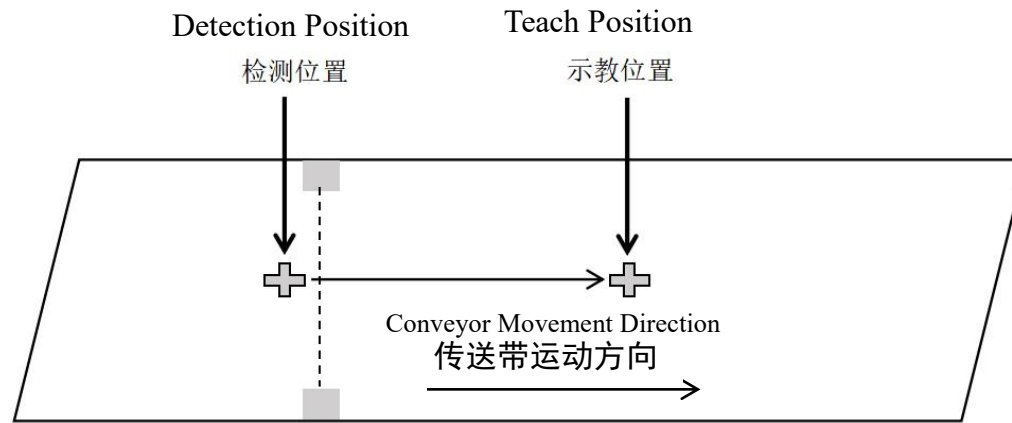
The purpose of sensor calibration is to determine the position of an object's reference point in the conveyor coordinate system at the moment the sensor is triggered. This reference point serves as the origin of the work object coordinate system and is usually defined as the pick position.

Operation method

Step 1: Adjust the conveyor speed to normal operating speed and place the object upstream of the sensor. Allow the object to move with the conveyor. When the object passes the sensor, the sensor is triggered. Stop the conveyor after the object enters the robot's workspace, ensuring the sensor is triggered only once during this process.

Step 2: Move the robot's tool end to align with the object's reference point and click "Read Teach Position". At this moment, the robot's position in the conveyor coordinate system represents the position of the object coordinate system.

Step 3: Click "Calculate Detection Position". The system will automatically compute the position of the object at the moment it triggered the sensor.



8.2.4.7. Vision Calibration – Vision Point Acquisition

There are two ways to acquire vision point data: manual input and communication retrieval.

A maximum of nine points and a minimum of three points are allowed. If all data of a point are zeros, the system considers that point as invalid. The points should ideally cover all areas within the camera's field of view.

Operation method

● Manual Input

Place the calibration plate on the conveyor belt under the camera. Capture an image of the calibration plate from the vision system to obtain the pixel coordinates of the nine calibration points. Click “Get Current Position” to read the conveyor position at the moment of image capture. Enter the pixel coordinates of each point, in order, into the corresponding fields on the “Vision Point Acquisition” interface.

● Communication Acquisition

Place the calibration plate on the conveyor belt under the camera. Capture an image of the calibration plate from the vision system to obtain the pixel coordinates of the nine calibration points. Click “Get Current Position” to read the conveyor position at the moment of image capture. Click “Communication Acquisition” to receive the pixel coordinate information of each calibration point.

The data transmission format for communication acquisition is as follows:

@X1,Y1,X2,Y2 ... X9,Y9\r

The start character is @, and the stop character is \r.

8.2.4.8. Vision Calibration – Teaching Point Acquisition

Operation method

Step 1: Maintain the same relative position between the calibration plate and the conveyor as in “Vision Expression” – “Vision Point Acquisition”. Move the conveyor until the calibration plate enters the robot’s working area, then stop the conveyor.

Step 2: Click “Get Current Position” to obtain the conveyor’s teaching position.

Step 3: Following the order of the vision pixel points, enter the point number in the “Robot Teaching Point” input box. Move the robot in the base coordinate system, align the robot’s end calibration cone with the corresponding calibration point, and click “Get Current Point” to record the point coordinates.

Step 4: Repeat “Step 3” until all calibration point coordinates are collected.

Step 5: Click “Calibrate/Download” to calculate the calibration results and save them.

8.2.5. Program Examples

8.2.5.1. Single Pick-and-Place for One Type of Object

The following is a program example for single pick-and-place when there is only one type of object on the conveyor belt:

Variable Declaration

```
1 Program ProgramBelt
2 Var
3 //Vision communication status
4 bVar1:Bool;
5 //Object state
6 bCanPick:Bool;
7 //grasping point
8 PickPose:RobotTarget:=((0,0,-20,0,0,0),(-1,0,-1,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
9 //Wait point
10 WaitPose:RobotTarget :=((500.8445,-41.9692,-10.3781,0,0,21.6027),(-1,0,2,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
11 //Place point
12 PlacePose:RobotTarget := ((400.8445,-41.9692,-10.3781,0,0,21.6027),(-1,0,2,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
13 //Intermediate variable
14 WaitJumpData:JumpData:=(10,20,-10,20,0,0);
15 PickJumpData:JumpData:=(-10,20,10,20,0,0);
16 EndVar
17 |
```

Program Editing

```

1 //Create communication between robots and vision
2 While bVar1 <> True Do
3   bVar1 := SocketCreate('192.168.100.100',5556,'Server1');
4 EndWhile
5 //Enable vision inspection
6 ConveyorVision(1,True,'Server1');
7 //The robot moves to the waiting point
8 MoveJump WaitPose, vel1, zone1,WaitJumpData, acc1;
9 Label1::
10 //Check if there is an object present
11 bCanPick := GetConveyorObject(1,1);
12 //If there are no objects, please continue searching
13 If False = bCanPick Then
14   Goto(Label1);
15 EndIf
16 //Bind the conveyor belt and switch to a dynamic coordinate system
17 TrackConveyor(1);
18 //The robot moves to the grasping point and grasps the object
19 MoveJump PickPose,vel2,zone1,PickJumpData,acc2;
20 //Robot follows object for 100ms
21 Wait(0.1);
22 //Unbind the conveyor belt
23 TrackBase;
24 //The robot moves to the placement point
25 MoveJump PlacePose,vel1,zone1,WaitJumpData, acc1;
26 Goto(Label1);
27 |

```

8.2.5.2. Single Pick-and-Place for Multiple Types of Objects

The following is a program example for single pick-and-place when there are four types of objects on the conveyor belt:

Note:

When multiple object types exist, they should be queried sequentially. As soon as a pickable target object is found, the pick operation should be executed immediately without continuing the subsequent queries.

Variable Declaration

```

1 Program Programbelt_objs
2 Var
3 //Vision communication status
4 bVar1 :Bool;
5 //Object state
6 bCanPick:Bool;
7 bCanPick2:Bool;
8 bCanPick3:Bool;
9 bCanPick4:Bool;
10 //grasping point
11 stPickPose:RobotTarget:=((0,0,-20,0,0,0),(-1,0,-1,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
12 //Wait point
13 WaitPose:RobotTarget:=((500.8445,-41.9692,-10.3781,0,0,21.6027),(-1,0,2,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
14 //Place point
15 PlacePose:RobotTarget:=((500.8445,-41.9692,-10.3781,0,0,21.6027),(-1,0,2,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
16 //Intermediate variable
17 WaitJumpData:JumpData:=(10,20,-10,20,0,0);
18 stPickJumpData:JumpData:=(-10,20,10,20,0,0);
19 EndVar
20 |

```

Program Editing


```

1 //Create communication between robots and vision
2 While bVar1 <> True Do
3   bVar1:=SocketCreate('192.168.100.100',5556,'Server1');
4 EndWhile
5 //Enable vision inspection
6 ConveyorVision(1,True,'Server1');
7 //The robot moves to the waiting point
8 MoveLJump WaitPose,vel1,zone1,WaitJumpData,acc1;
9 Label1::
10 //Check if there is an object present
11 bCanPick:=GetConveyorObject(1,1);
12 If bCanPick=True Then
13   Goto(Pick);
14 EndIf
15 bCanPick2:=GetConveyorObject(1,2);
16 If bCanPick2=True Then
17   Goto(Pick);
18 EndIf
19 bCanPick3:=GetConveyorObject(1,3);
20 If bCanPick3=True Then
21   Goto(Pick);
22 EndIf
23 bCanPick4:=GetConveyorObject(1,4);
24 If bCanPick4=True Then
25   Goto(Pick);
26 EndIf
27 //If there are no objects, please continue searching
28 If False = bCanPick And False = bCanPick2 And False = bCanPick3 And False = bCanPick4 Then
29   Goto(Label1);
30 EndIf
31 Pick::
32 //Bind the conveyor belt and switch to a dynamic coordinate system
33 TrackConveyor(1);
34 //The robot moves to the grasping point and grasps the object
35 MoveLJump stPickPose,vel2,zone1,stPickJumpData,acc2;
36 //Robot follows object for 100ms
37 Wait(0.1);
38 //Unbind the conveyor belt
39 TrackBase;
40 //The robot moves to the placement point
41 MoveLJump PlacePose,vel1,zone1,WaitJumpData,acc1;
42 Goto(Label1);

```

8.2.6. Frequently Asked Questions (FAQ)

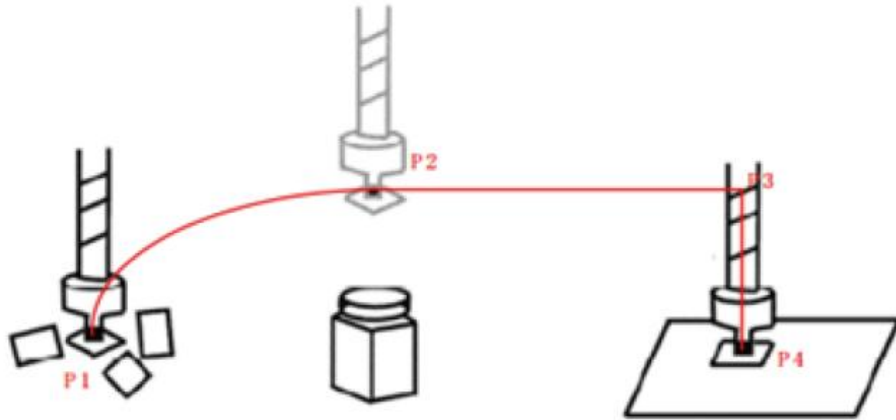
8.3. Fly Camera

8.3.1. Overview

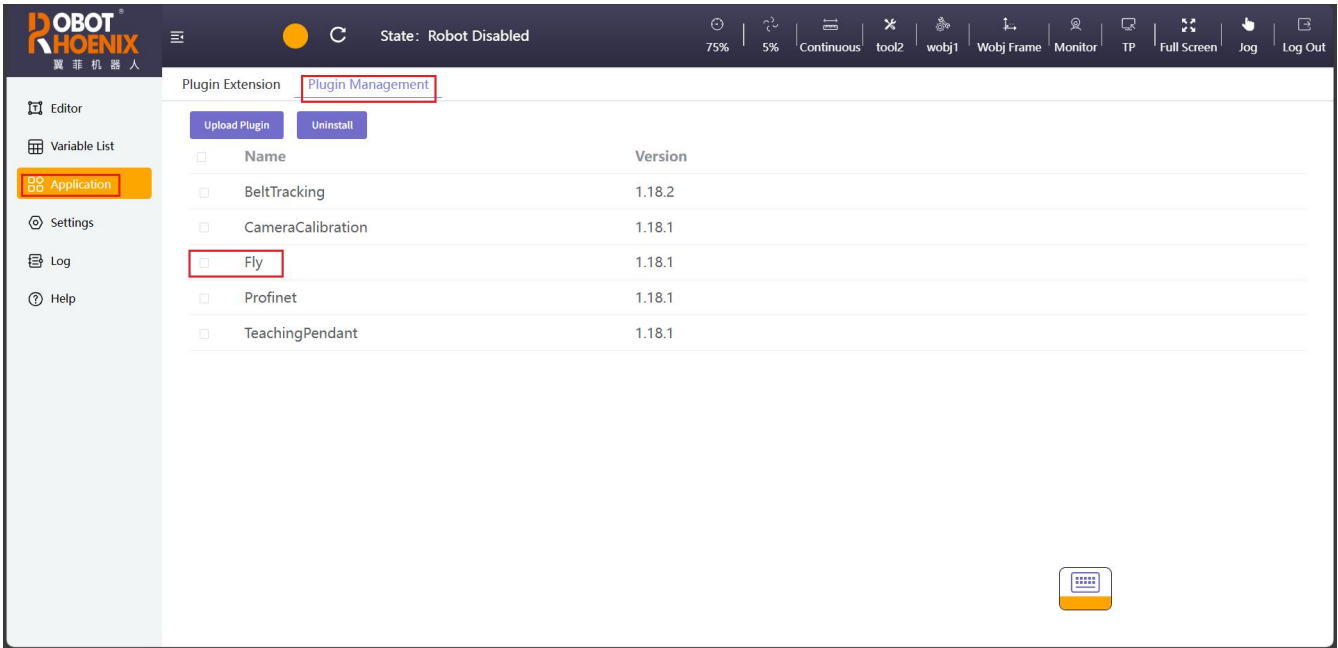
Fly camera refers to a process where the robot completes image capture and position adjustment for the placement point without stopping its motion from the pick-up point to the placement point. This reduces cycle time and increases operational efficiency.

As shown in the diagram below, P1 is the pick-up point, P2 is the image capture point, P3 is the transition point, and P4 is the placement point. Under normal conditions, the robot pauses at P1, P2, P3, and P4. However, during fly camera, the robot does not stop at P2. Instead, as the robot moves from P1 to P3, it predicts when it is approaching P2 and triggers a DO (Digital

Output) signal to initiate camera image capture and save position data. Based on the vision processing results and the saved position data, the placement position and orientation at P4 are determined.



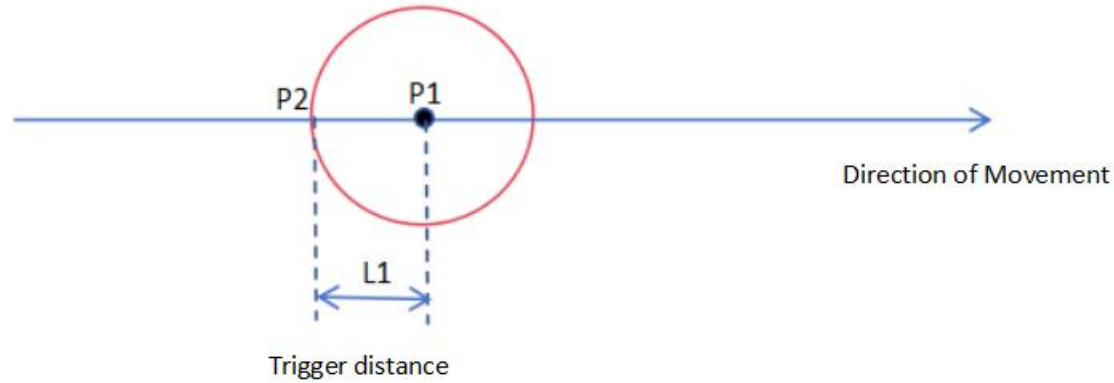
Before using the fly camera functionality in the Plugin Extension, upload the "Fly" and "CameraCalibration" plugins via Application> Plugin Management > Upload Plugin, and calibrate the required vision coordinate system.



8.3.2. Basic Parameters

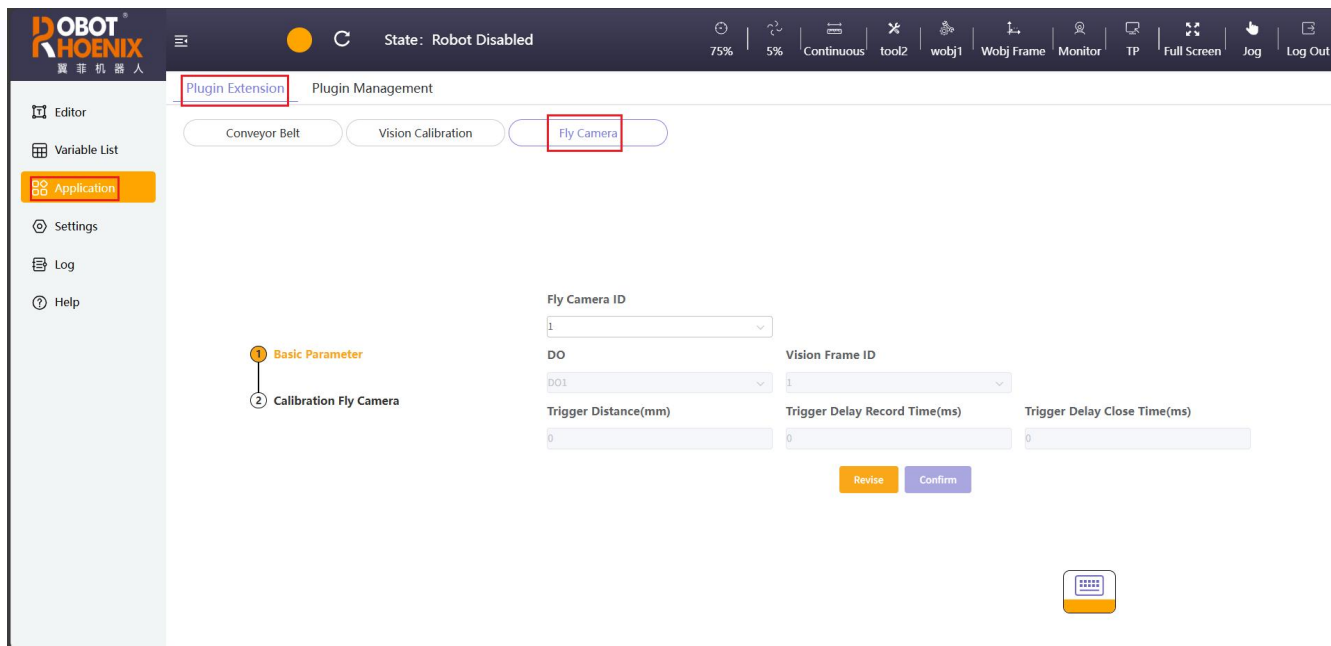
As shown in the figure below, P1 is the taught shooting point, which corresponds to the “Robot Coordinate of the Shooting Point” in Fly-shoot Calibration. L1 is the trigger distance. When the robot moves into the spherical zone centered at P1 with a radius of L1, the bound DO signal is set HIGH to trigger the camera to capture an image. After the configured

capture-delay time elapses, the robot's position is recorded. After the configured signal-off delay time, the DO signal used to trigger the camera is reset.



Operation Method

Click “Application” → “Plugin Extension” → “Fly Camera” → “Basic Parameters” in the left navigation panel to enter the Fly Camera Basic Parameter Settings interface, as shown below.



- Fly Camera ID

Click to expand the “Fly Camera ID” dropdown list and select an ID. The allowable range is 1 to 16.

- DO

The DO signal used to trigger the camera. When the robot approaches the shooting point

(based on the configured trigger distance), the DO is set HIGH to trigger the camera. The allowable range is 1 to 16.

- Vision Coordinate System ID

The calibrated vision coordinate system. Click to expand the “Vision Coordinate System ID” dropdown list and select an ID. The allowable range is 1 to 16.

- Trigger Distance

When the distance between the robot and the shooting point is less than or equal to the “Trigger Distance,” the DO used to trigger the camera is set HIGH.

Unit: mm; allowable range: 0 to 9999.

- Trigger Delay – Position Recording

After the DO is set HIGH (i.e., the camera is triggered), the system waits for a configured delay before recording the robot's current position.

Unit: ms; allowable range: 0 to 9999.

- Trigger Delay – DO Reset

After the DO is set HIGH (i.e., the camera is triggered), the system waits for a configured delay before resetting the DO signal.

Unit: ms; allowable range: 0 to 9999.

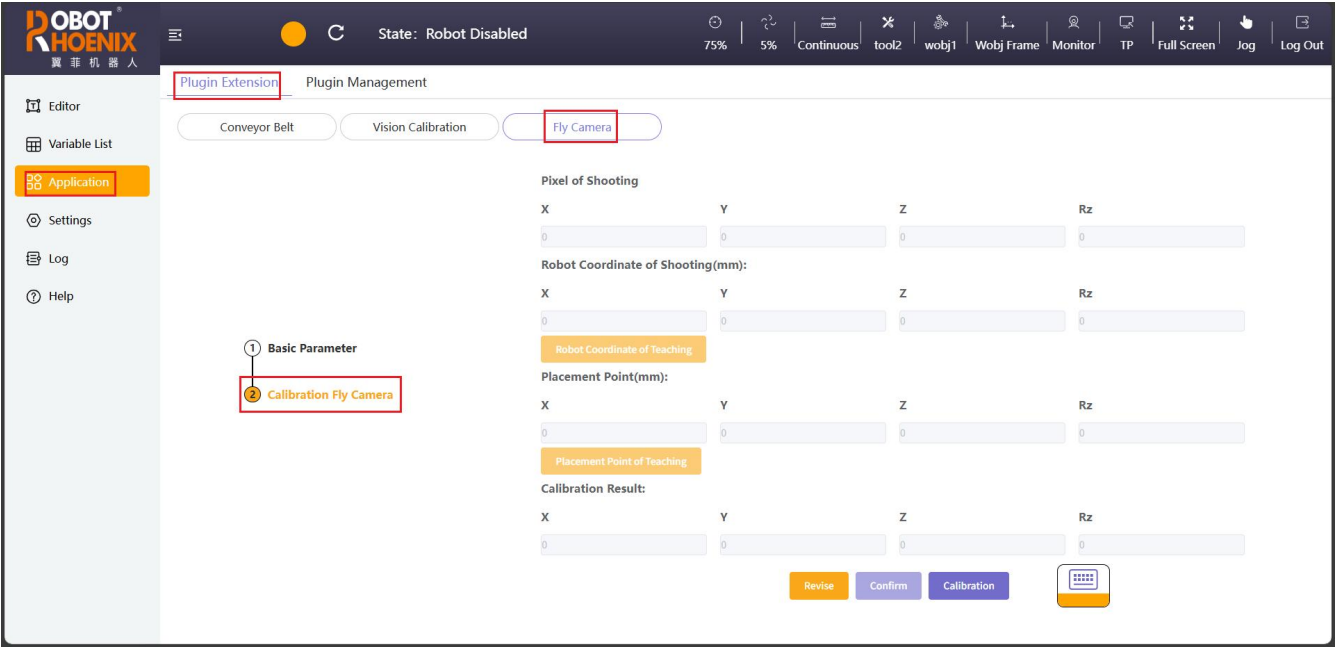
8.3.3. Fly Camera Calibration

Operation Method

Step 1: Click “Application” → “Plugin Extension” → “Fly-shoot” → “Basic Settings” in the left navigation panel to enter the Fly Camera Basic Parameter Settings interface. Click “Edit”, then select the fly camera ID, the calibrated Vision Coordinate System ID, the DO signal used to trigger the camera, the trigger distance to the shooting point, the trigger-delay time for position recording, and the trigger-delay time for DO reset. Click “OK” to confirm.

Step 2: Click “Application” → “Plugin Extension” → “Fly Camera” → “Fly Camera Calibration” in the left navigation panel to enter the Fly-shoot Calibration interface. Move the robot to the placement point, click the “Edit” button, then click “Teach Placement Point” to obtain the robot coordinates of the placement point. At the same time, a variable of type RobotTarget needs to be declared in the fly camera program, and this point must be taught.
(rPlaceTarget: RobotTarget; // Robot coordinates of the placement point during teaching)

Step 3: After completing the above steps, move the robot to pick the material from the taught placement point, then move it to the shooting point. Click “Teach Robot Coordinates” to obtain the current robot coordinates and pixel coordinates.



Step 4: Click “Confirm”.

Step 5: Click “Calibrate” to complete the process.

8.3.4. Program Example

The following is an example of a Fly-shoot program:

Variable Declaration

```
2  var
3  //FlyingID
4  iFlyId: Int:=1;
5  //Get Socket Communication Status
6  bCreateStatus: Bool;
7  //Flying status
8  bCheckData: Bool;
9  //Pixel coordinates of objects in visual recognition
10 pVisionPose: Pose;
11 //Robot coordinates during photography
12 pRobPose: Pose;
13 //String read through Socket communication
14 sReceiveMessage: String:='50,60,70;';
15 //Find the position of the first comma
16 iFindFirstComma: Int;
17 //Find the position of the second comma
18 iFindSecondComma: Int;
19 //Find the position of the semicolon
20 iFindSemi: Int;
21 //Expressing X coordinate with string type
22 sPoseX: String;
23 //Expressing Y coordinate with string type
24 sPoseY: String;
25 //Expressing RZ coordinate with string type
26 sPoseRZ: String;
27 //Delete the remaining characters before the first comma
28 sDeleteFirstComma: String;
29 //Delete the remaining characters before the second comma
30 sDeleteSecondComma: String;
31 //Calculate the coordinates of the placement point
32 rCalTarget: RobotTarget;
33 //Coordinates of the placement point during teaching
34 rPlaceTarget: RobotTarget;
35 //Fetching point
36 P_FETCH: RobotTarget:=((319.1253,267.6081,-70,0,0,68.2111),(0,0,13,-1,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
37 //Waiting point
38 P_WAIT: RobotTarget:=((313.9253,267.6081,-70,0,0,68.2111),(0,0,13,-1,0,0,0,2),(0,0,0,0,0,0),tool1,wobj1);
39 p1: RobotTarget:=((483.1985,-170.1114,0,0,0,-42.8731),(0,-1,9,0,0,0,0,0),(0,0,0,0,0,0),tool1,wobj1);
40 //Photo point
41 p2: RobotTarget:=((483.1985,-170.1114,-50,0,0,-42.8731),(0,-1,9,0,0,0,0,1),(0,0,0,0,0,0),tool1,wobj1);
42 //jumpdata
43 jumpdata2: JumpData[1]:=[(50,0,-50,0,50,-50)];
44 EndVar
```

Program Editing

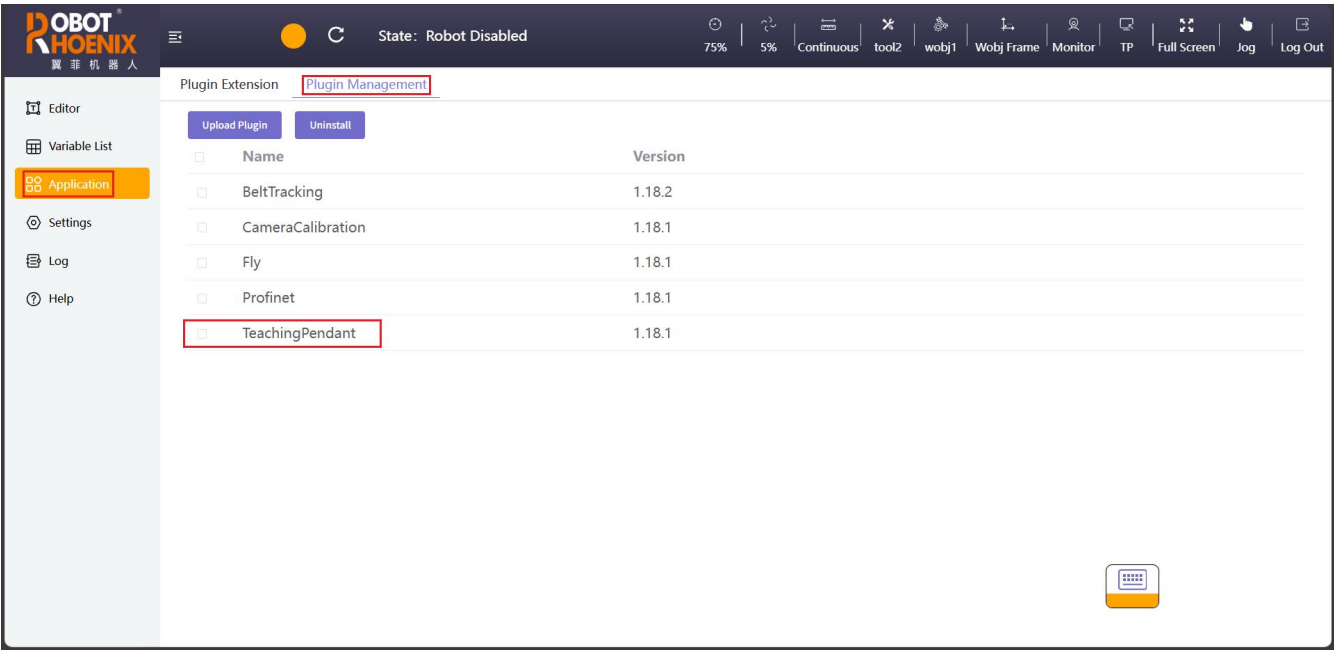
```

1 //Create communication between robots and vision
2 While True Do
3     //Create connection
4     bCreateStatus:=SocketCreate('192.168.100.100',4444,'Server1');
5     //Check the status, successfully created and exited the loop
6     If bCreateStatus =True Then
7         Exit;
8     Else
9         Wait(0.5);
10    EndIf
11 EndWhile
12 //Go to the material fetching location to fetch the material
13 MoveLJump P_FETCH,vell,zonel,jumpdata2[0],acc1;
14 //Setup flying data
15 SetFly(iFlyId);
16 label1::
17 //Go to the starting point
18 MoveLJump p1,vell,zonel,jumpdata2[0],acc1;
19 //Start flying
20 StartFly();
21 //Move to the set range and start taking photos
22 MoveL p2,vell,zonel,acc1;
23 //End the aerial photography, and all subsequent commands will be regular commands
24 StopFly();
25 //The inspection of the flying has been successful
26 bCheckData:=CheckFly(iFlyId);
27 //If successful, continue and further obtain the location. If unsuccessful, go to the waiting point
28 If bCheckData Then
29     //Obtain robot coordinates when taking photos
30     pRobPose:=GetFlyRobotPose(iFlyId);
31     //Obtain the pixel coordinates of the object during photography
32     sReceiveMessage:=SocketReadString('Server1');
33     //Analysis
34     If sReceiveMessage <>' 'Then
35         //Find the position of the first comma
36         //Obtain the x-coordinate
37         iFindFirstComma := StrFind(sReceiveMessage,',');
38         sPoseX :=StrMid(sReceiveMessage,iFindFirstComma, 0);
39         sDeleteFirstComma := StrDelete(sReceiveMessage, iFindFirstComma+1, 0);
40         //Find the position of the second comma
41         //Obtain the y-coordinate
42         iFindSecondComma := StrFind(sDeleteFirstComma,',');
43         sPoseY := StrMid(sDeleteFirstComma, iFindSecondComma, 0);
44         sDeleteSecondComma := StrDelete(sDeleteFirstComma,iFindSecondComma+1,0);
45         //Find the position of the semicolon
46         //Obtain the Rz-coordinate
47         iFindSemi:=StrFind(sDeleteSecondComma,';');
48         sPoseRZ := StrMid(sDeleteSecondComma,iFindSemi,0);
49         //Convert object pixel coordinates to Double type
50         pVisionPose.x:= StrToDouble(sPoseX);
51         pVisionPose.y := StrToDouble(sPoseY);
52         pVisionPose.rz := StrToDouble(sPoseRZ);
53     EndIf
54 //Adjust the position of the object for correction
55 rCallTarget:=CalculationFlyResult(iFlyId, 2,pRobPose, pVisionPose,rPlaceTarget);
56 //Go to the new feeding point
57 MoveL rCallTarget,vell,zonel,acc1,tool2;
58 Goto(label1);
59 Else
60     //Go to the Waiting point
61     MoveLJump P_WAIT,vell,zonel,jumpdata2[0];
62 EndIf

```

9. Teaching Pendant

Before enabling the teaching pendant, the TeachingPendant plugin must first be uploaded via “Application- Plugin Management -Upload Plugin”. Refer to the figure below:



9.1. Teaching Pendant Appearance

The teaching pendant is equipped with operational keys and buttons necessary for robot teaching and programming, including touchscreen controls.



9.1.1. Mode Switch

There are two modes: When the knob is rotated to the left, it is in Manual Mode. When the knob is rotated to the right, it is in Auto Mode.

9.1.2. Emergency Stop Button

Pressing this button cuts off the servo power, stopping the system immediately. The motors are disabled, and if the motors have a brake function, they will engage.

9.1.3. Safety Switch

In Manual Mode, pressing the middle position of the safety switch enables the system.

9.1.4. Axis Operation Keys

The axis operation keys (also known as jog buttons) control the movement of the robot and external axes.

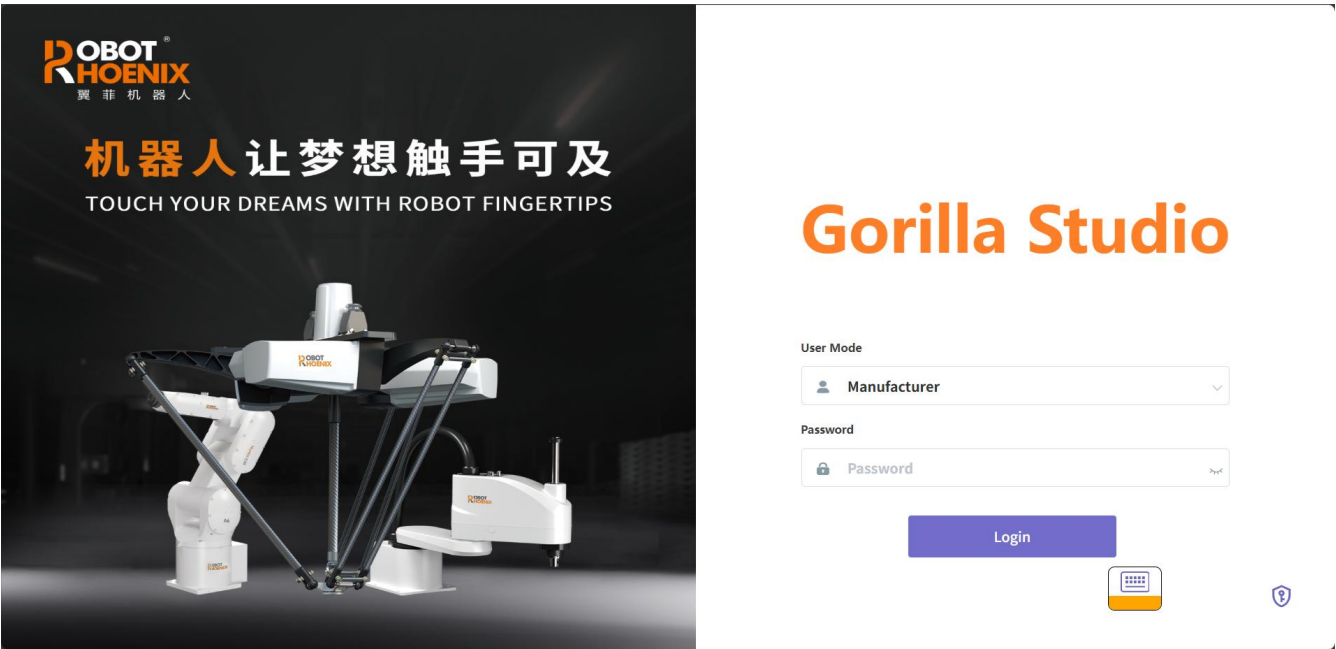
Note:

The axis operation keys are only functional in Manual Mode and when the program is not running.

9.2. Teaching Pendant Interface

9.2.1. Login Interface

After powering on the teaching pendant, the login interface appears, prompting the user to log in with a username and password. Refer to the figure below :



The robot system is configured with a default username and initial password at the factory. It is recommended to change and save the password after logging in.

Default Username and Initial Password:

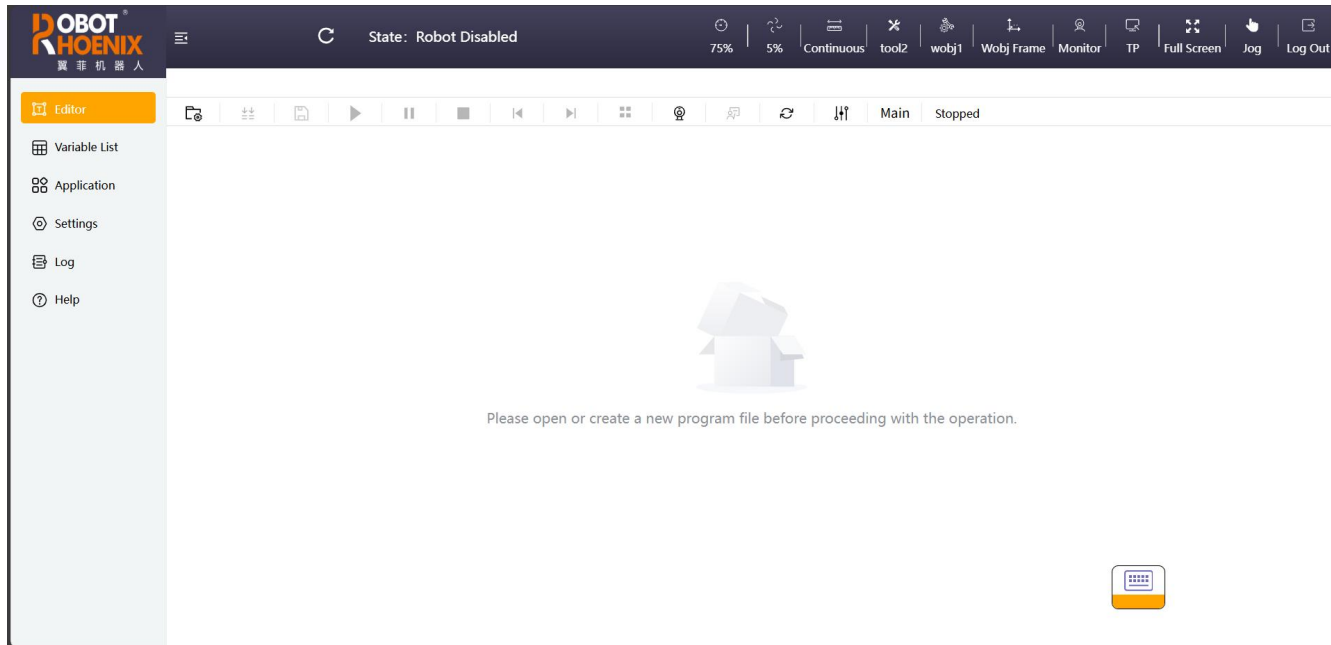
Username	Password
Administrator	123456
Programmer	123456
Operator	123456

Note:

To enter the authorization code, click the  button at the bottom-right corner of the login page.

9.2.2. Main Interface

After logging in, the system enters the main interface, as shown below.



The interface consists of the following parts:

- (1) Navigation Bar
- (2) Toolbar
- (3) Editing Area
- (4) Shortcut Commands
- (5) Compilation Messages
- (6) Virtual Keyboard

9.2.3. Physical Hardware Buttons

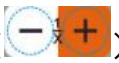
The layout of the physical hardware buttons on the teaching pendant is shown below.



Functions of the Left-Side Buttons:

- Start Button () : Starts the program.
- Stop Button () : Stops the program.
- Reset Button () : Clears system faults and resets the robot state.
- Enable Button () : Enables or disables the system in **Auto Mode**.
- Settings Button () : Opens the settings interface.
- Coordinate System Switch Button () : Switches the coordinate system.

Functions of the Right-Side Buttons:

- Jog Buttons () : Control robot or external axis movements.

Functions of the Bottom Buttons:

- Fn Button () : Switches single-step execution mode to "Previous Step" or "Next Step."
- Step Button () : Executes the program in single-step mode.
- V-Button () : Decreases motion speed. In **Manual Mode**, it reduces the jog speed; in **Auto Mode**, it decreases the automatic operation speed.
- V+Button () : Increases motion speed. In **Manual Mode**, it increases the jog speed; in **Auto Mode**, it increases the automatic operation speed.
- Pause Button () : Pauses program execution.

10. Appendix

10.1. Modbus Slave Address Table

Address	Data type	Content	Note
Master communication attribute: Read discrete inputs, function code 0x02.			
0	Bit	Program running status: (0-Stopped, 1-Running)	
1	Bit	Program Pause Status (0-Not Paused, 1-Paused)	
2	Bit	Program Error Status (0-No Error, 1-Error Present)	
3	Bit	Emergency Stop Status (0-OFF, 1-ON)	
4	Bit	Enable Status (0-OFF, 1-ON)	
...	Bit	Reserved	
16	Bit	RobotTarget variable write success (0-failure, 1-success)	
17	Bit	RobotTarget variable read success (0-failure, 1-success)	
18	Bit	Point-to-point motion completion status (0-invalid or not in position, 1-completed)	
...	Bit	Reserved	
999	Bit		
Master communication attributes: Read/write coils, function codes: 0x01, 0x05, 0x0F.			
0	Bit	Program start/run / Resume pause	(Return-to-zero type)
1	Bit	Program pause	(Return-to-zero type)
2	Bit	Program stop	(Return-to-zero type)
3	Bit	Fault reset	(Return-to-zero type)
4	Bit	Enable Switch (0-OFF,1-ON)	
...	Bit	Reserved	
20	Bit	Program select 1	
21	Bit	Program select 2	
22	Bit	Program select 3	
23	Bit	Program select 4	
24	Bit	Program select 5	
25	Bit	Program select 6	

26	Bit	Program select 7	
27	Bit	Program select 8	
28	Bit	Program select 9	
29	Bit	Program select 10	
30	Bit	Program select 11	
31	Bit	Program select 12	
32	Bit	Program select 13	
33	Bit	Program select 14	
34	Bit	Program select 15	
35	Bit	Program select 16	
36	Bit	Program select 17	
37	Bit	Program select 18	
38	Bit	Program select 19	
39	Bit	Program select 20	
...	Bit	Reserved	
48	Bit	Jog J1/X +	(Return-to-zero type)
49	Bit	Jog J2/Y +	(Return-to-zero type)
50	Bit	Jog J3/Z +	(Return-to-zero type)
51	Bit	Jog J4/RZ +	(Return-to-zero type)
52	Bit	Jog J5/RX +	(Return-to-zero type)
53	Bit	Jog J6/RX +	(Return-to-zero type)
54	Bit	Reserved	
55	Bit	Reserved	
56	Bit	Jog J1/X -	(Return-to-zero type)
57	Bit	Jog J2/Y -	(Return-to-zero type)
58	Bit	Jog J3/Z -	(Return-to-zero type)
59	Bit	Jog J4/RZ -	(Return-to-zero type)
60	Bit	Jog J5/RX -	(Return-to-zero type)
61	Bit	Jog J6/RX -	(Return-to-zero type)
62	Bit	Reserved	
63	Bit	Reserved	

64	Bit	Write the robot's current position into the current RobotTarget variable	(Return-to-zero type)
65	Bit	Write the modified position (such as holding register address 17) into the current RobotTarget variable	(Return-to-zero type)
66	Bit	Move point-to-point to the position of the current RobotTarget variable	(Return-to-zero type)
...	B	Reserved	
999	Bit		
Master communication attribute: Read input registers, function code: 0x04.			
0	Word	Robot status codes: 0 - Robot not configured 1 - Robot not enabled 2 - Robot enabled 3 - Robot homing in progress 4 - Robot moving 5 - Robot stopping 6 - Robot fault 7 - Robot jogging 9 - Robot emergency stop 100 - System not connected	
1	Word	Alarm codes: See the help document for details.	
2	Word	Current Auto Speed	
3	Word	Current Jog Speed	
4	Word	Current Coordinate System	
...	Word	Reserved	
20	Float	Robot X coordinate.	
21			
22	Float	Robot Y coordinate.	
23			
24	Float	Robot Z coordinate.	
25			
26	Float	Robot RX coordinate.	
27			
28	Float	Robot RY coordinate.	
29			
30	Float	Robot RZ coordinate.	
31			
...	Word	Reserved	
40	Float	Robot 1 Axis position	
41			
42	Float	Robot 2 Axis position	

43			
44	Float	Robot 3 Axis position	
45			
46	Float	Robot 4 Axis position	
47			
48	Float	Robot 5 Axis position	
49			
50	Float	Robot 6 Axis position	
51			
...	Word	Reserved	
60	Float	Current X coordinate of the RobotTarget variable	
61			
62	Float	Current Y coordinate of the RobotTarget variable	
63			
64	Float	Current Z coordinate of the RobotTarget variable	
65			
66	Float	Current RX coordinate of the RobotTarget variable	
67			
68	Float	Current RY coordinate of the RobotTarget variable	
69			
70	Float	Current RZ coordinate of the RobotTarget variable	
71			
72	Word(Signed)	Current quadrant of Axis 1 rotation of the RobotTarget variable	
73	Word(Signed)	Current quadrant of Axis 2 rotation of the RobotTarget variable	
74	Word(Signed)	Current quadrant of Axis 3 rotation of the RobotTarget variable	
75	Word(Signed)	Current quadrant of Axis 4 rotation of the RobotTarget variable	
76	Word(Signed)	Current quadrant of Axis 5 rotation of the RobotTarget variable	
77	Word(Signed)	Current quadrant of Axis 6 rotation of the RobotTarget variable	
78	Word(Signed)	Current quadrant of Axis 7 rotation of the RobotTarget variable	
79	Word	Pose type written to the RobotTarget variable	
80	Word	Current tool coordinate system number of the RobotTarget variable	
81	Word	Current work object coordinate system number of the RobotTarget variable	
...	Word	Reserved	

999	Word		
Master communication attributes: Read/write holding registers, function codes: 0x03, 0x10.			
0	Word	Auto speed setting (1–100)	
1	Word	Jog speed setting (1–100)	
2	Word	Jog mode selection (0–Continuous, 1–Step)	
3	Float	Step length (in joint coordinates: degrees; in Cartesian coordinates: millimeters)	
4			
5	Word	Coordinate system selection (0–Joint, 1–Base, 2–World, 3–Tool, 4–User, 5–Work object)	
6	Word	Point-to-point motion mode setting (0–MoveJ, 1–MoveL, 2–MoveLJump)	
...	Word	Reserved	
16	Word	Index of the current RobotTarget variable	
17	Float	X coordinate written to the RobotTarget variable	
18			
19	Float	Y coordinate written to the RobotTarget variable	
20			
21	Float	Z coordinate written to the RobotTarget variable	
22			
23	Float	RX coordinate written to the RobotTarget variable	
24			
25	Float	RY coordinate written to the RobotTarget variable	
26			
27	Float	RZ coordinate written to the RobotTarget variable	
28			
29	Word(Signed)	Rotation quadrant of axis 1 written to the RobotTarget variable	
30	Word(Signed)	Rotation quadrant of axis 2 written to the RobotTarget variable	
31	Word(Signed)	Rotation quadrant of axis 3 written to the RobotTarget variable	
32	Word(Signed)	Rotation quadrant of axis 4 written to the RobotTarget variable	
33	Word(Signed)	Rotation quadrant of axis 5 written to the RobotTarget variable	
34	Word(Signed)	Rotation quadrant of axis 6 written to the RobotTarget variable	
35	Word(Signed)	Rotation quadrant of axis 7 written to the RobotTarget variable	
36	Word	Pose type written to the RobotTarget	

		variable	
37	Word	Tool coordinate system number written to the RobotTarget variable	
38	Word	Work object coordinate system number written to the RobotTarget variable	
...	Word	Reserved	
48	Float	Gantry motion upward distance (-999-999)	
49			
50	Float	Gantry motion upward turning-zone ratio (0-100)	
51			
52	Float	Gantry motion downward distance (-999-999)	
53			
54	Float	Gantry motion downward turning-zone ratio (0-100)	
55			
56	Float	Gantry Motion Posture Start Height (-999 - -999)	
57			
58	Float	Gantry Motion Posture Stop Height (-999 - -999)	
59			
...	Word	Reserved	
999	Word		

10.2. PLC Address Table

Address	Data type	Content		Note
Input				
128	Byte	Bit0	Program Start / Resume Pause	（Return-to-zero type）
		Bit1	Program pause	（Return-to-zero type）
		Bit2	Program stop	（Return-to-zero type）
		Bit3	Fault reset	（Return-to-zero type）
		Bit4	Enable Switch（0-OFF,1-ON）	
		...	Reserved	
		Bit7		
129	Byte	Program Select（1-20）		

Address	Data type	Content		Note
Output				
128	Byte	Bit0	Program Running Status (0 – Stop, 1 – Run)	
		Bit1	Program Pause Status (0 – Not Paused, 1 – Paused)	
		Bit2	Program Error Status (0 – No Error, 1 – Error)	
		Bit3	Emergency Stop Status (0 – OFF, 1 – ON)	
		Bit4	Enable Status (0 – OFF, 1 – ON)	
		...	Reserved	
		Bit7		

10.3. SCARA 4-Axis Torque Homing Instructions

Note: Please perform homing strictly according to this instruction; otherwise, homing may be inaccurate!

10.3.1. Hardware Preparation

Limit block, L-shaped homing fixture, and three screws (one M4*10 and two M4*15).

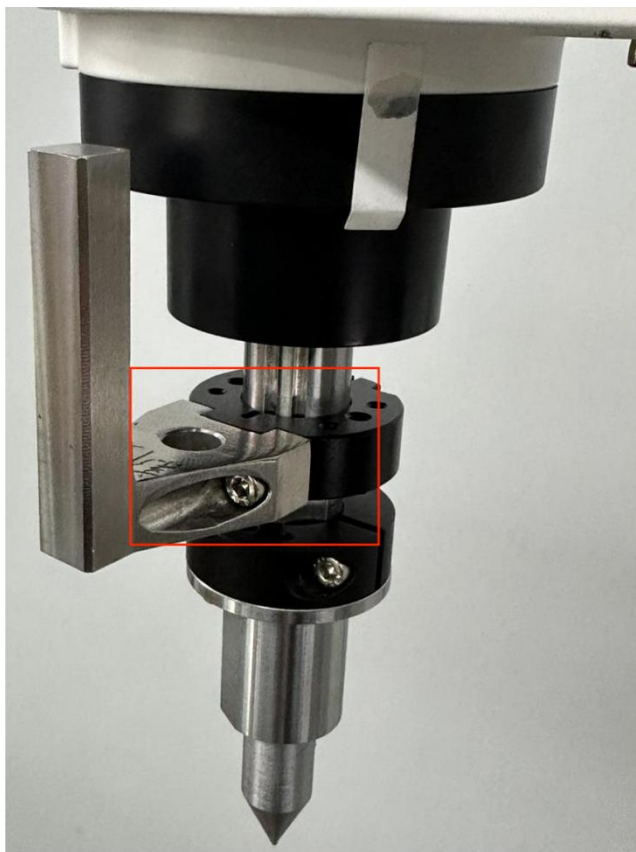
10.3.2. Installation

Operation Method

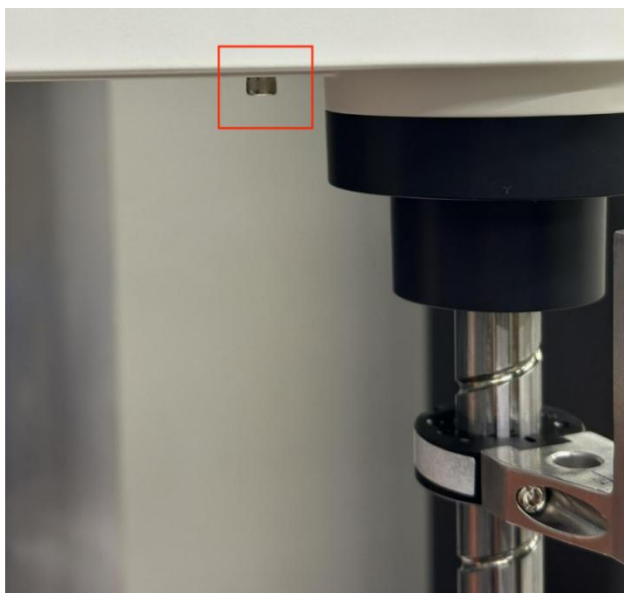
Step 1: Install the Limit Block. Mount the limit block at a specific position on the lead screw.

Note: After homing is completed, the limit block must not be removed. Carefully select the installation position.

Step 2: Install the Homing Fixture. Fix the L-shaped homing fixture to the limit block using two screws (M4*15), as shown in the figure below:



Step 3: Install the screw (M4*10) at the designated position on the robot's small arm, as shown below.



10.3.3. Homing

Operation Method

Step 1: Homing Position.

Manually push the homing fixture to the right side until it aligns with the limit screw, and ensure the top surface aligns with the protrusion of the second axis (as shown in the figure below). While in the enable state, jog the robot to move downward along the Z-axis in 2 mm increments.

Note: If homing has previously been performed using a different procedure than this one, please follow the original homing position during subsequent homing operations to ensure consistency of the zero positions for all four axes.

**Step 2: Jog the Fourth Axis.**

With the robot in the enable state, jog the fourth axis to rotate it 20° counterclockwise, as shown in the figure below:

**Step 3:** Homing.

Enter the robot zero calibration interface, select the homing method as 'Torque Homing', and click the 'Calibrate' button corresponding to A4 to start the homing process.

Step 4: History Record.

After homing is completed, click the 'History' button to view the homing encoder values.