



DOBOT

User Guide

DobotSCStudio

User Guide

Issue: V1.0

Date: 2020-06-03

Shenzhen Yuejiang Technology Co., Ltd

Copyright © Shenzhen Yuejiang Technology Co., Ltd 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without the prior written consent of Yuejiang Technology Co., Ltd

Disclaimer

To the maximum extent permitted by applicable law, the products described (including its hardware, software, and firmware, etc.) in this document are provided **AS IS**, which may have flaws, errors or faults. Yuejiang makes no warranties of any kind, express or implied, including but not limited to, merchantability, satisfaction of quality, fitness for a particular purpose and non-infringement of third party rights. In no event will Yuejiang be liable for any special, incidental, consequential or indirect damages resulting from the use of our products and documents.

Before using our product, please thoroughly read and understand the contents of this document and related technical documents that are published online, to ensure that the robot is used on the premise of fully understanding the robot and related knowledge. Please use this document with technical guidance from professionals. Even if follow this document or any other related instructions, Damages or losses will be happening in the using process, Dobot shall not be considered as a guarantee regarding all security information contained in this document.

The user has the responsibility to make sure following the relevant practical laws and regulations of the country, in order that there is no significant danger in the use of the robot.

Shenzhen Yuejiang Technology Co., Ltd

Address: Address: Floor 9-10, Building 2, Chongwen Garden, Nanshan iPark, Liuxian Blvd,
Nanshan District, Shenzhen, Guangdong Province, China

Website: www.dobot.cc

Preface

Purpose

This manual introduces the functions and usage of the robot control software DobotSCStudio, which is convenient for users to understand and use robot.

Intended Audience

This document is intended for:





- Customer
- Sales Engineer
- Installation and Commissioning Engineer
- Technical Support Engineer

Change History

Date	Change Description
2020/06/03	The first release

Symbol Conventions

The symbols that may be founded in this document are defined as follows.

Symbol	Description
 DANGER	Indicates a hazard with a high level of risk which, if not avoided, could result in death or serious injury
 WARNING	Indicates a hazard with a medium level or low level of risk which, if not avoided, could result in minor or moderate injury, robot damage
 NOTICE	Indicates a potentially hazardous situation which, if not avoided, can result in equipment damage, data loss, or unanticipated result
 NOTE	Provides additional information to emphasize or supplement important points in the main text

Contents

1. Function Description	1
1.1 Overview.....	1
1.1.1 Main Interface Description.....	1
1.2 Settings.....	2
1.2.1 Setting Motion Parameter.....	2
1.2.2 Setting User Coordinate System.....	6
1.2.3 Setting Tool Coordinate System.....	9
1.2.4 Homing.....	14
1.2.5 Calibration.....	20
1.2.6 VirtualRobot.....	22
1.2.7 Log.....	22
1.2.8 Language.....	23
1.2.9 Network Service.....	23
1.3 Monitor.....	25
1.3.1 I/O Monitor.....	25
1.4 Remote Control.....	26
1.4.1 Remote I/O.....	26
1.4.2 Remote Modbus.....	28
1.5 Programming.....	29
1.5.1 Project Description.....	29
1.5.2 Programming Panel Description.....	30
1.5.3 Programming Description.....	31
1.6 Enabling.....	41
1.7 Setting Global Velocity Rate.....	41
1.8 Alarm Description.....	42
2. Program Language	44
2.1 Arithmetic Operators.....	44
2.2 Relational Operator.....	44
2.3 Logical Operators.....	44
2.4 General Keywords.....	45
2.5 General Symbol.....	45
2.6 Processing Control Commands.....	45
2.7 Global Variable.....	45
2.8 Motion Commands.....	46
2.9 Motion Parameter Commands.....	55
2.10 Input/output Commands.....	58
2.11 Program Managing Commands.....	59
2.12 Pose Getting Command.....	62
2.13 TCP.....	63
2.14 UDP.....	67
2.15 Modbus.....	71
2.15.1 Modbus Register Description.....	71

2.15.2	Command Description	72
2.16	ECP	75
2.17	Process Command.....	77
2.17.1	Conveyor Tracking Command.....	77
2.17.2	Pallet Commands.....	79
3.	Process Guide	85
3.1	Conveyor Tracking.....	85
3.1.1	Overview	85
3.1.2	Building Environment	85
3.1.3	Calibrating Conveyor	87
3.1.4	Configuring Conveyor.....	91
3.1.5	Example.....	103
3.2	Palletizing	107
3.2.1	Overview	107
3.2.2	Setting Pallet.....	108
3.2.3	Example.....	112
4.	Typical Applications	113
4.1	Modbus Application.....	113
4.2	I/O Application	115
4.2.1	Grabbing Bottle Application.....	115

1. Function Description

1.1 Overview

A SC series controller is equipped with DobotSCStudio, providing secondary development and various kinematic algorithms for mechanical structures, which are suitable for various applications.

1.1.1 Main Interface Description

Figure 1.1 shows the main interface of DobotSCStudio, Table 1.1 lists the interface description.

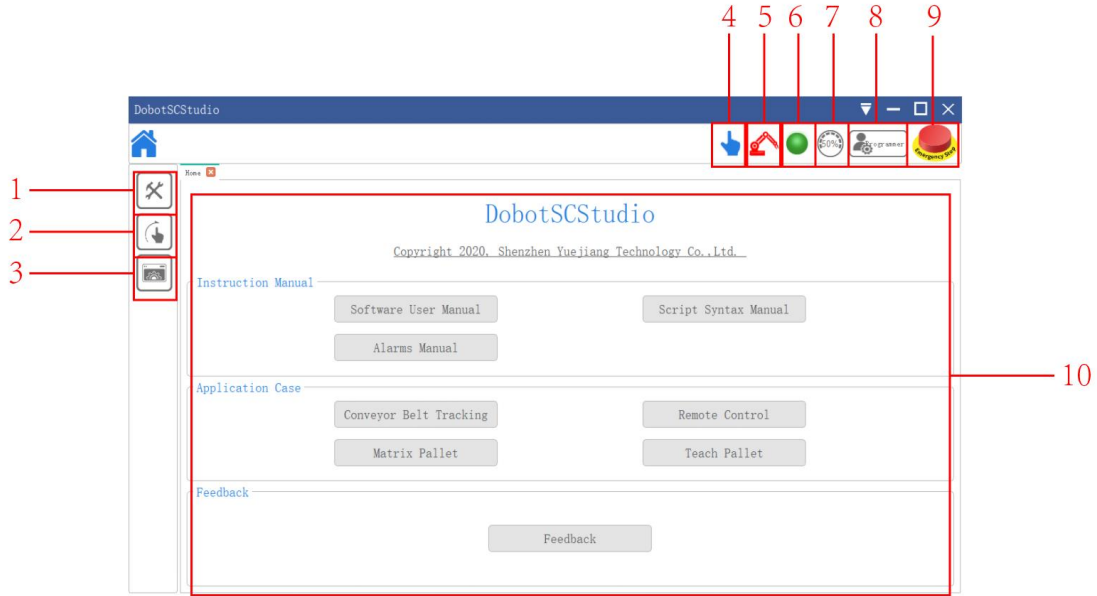




Figure 1.1 Main interface

Table 1.1 Interface description

No.	Description
1	Project You can build or import a project, and debug or run it
2	Jog Jog the robot in different coordinate systems. This function is valid only when DobotSCStudio is set to the manual mode Jog the robot in the Joint coordinate system: From top to bottom, jog J1, J2, ..., and J6 Jog the robot in the Cartesian coordinate system: From top to bottom, jog the X, Y, Z, R(A-axis), B, and C
3	System

No.	Description
	You can set system configurations. Such as NetworkSetting, RobotParams, Coordinate, Process
4	<p>You can click the icon to change manual mode and auto mode.</p> <ul style="list-style-type: none">  : In manual mode, indicate the motor status (enabled or disabled)  : In auto mode, indicate that you can click this button to control the motor
5	<ul style="list-style-type: none"> In manual mode, indicate the motor status (enabled or disabled) In auto mode, indicate that you can click this button to control the motor
6	<p>Check robot alarm</p> <p>When an alarm is triggered, this icon will turn red</p> <p>You can check the alarm details on the operation panel and clear it in the manual mode</p>
7	Set global velocity rate
8	<p>Select user mode</p> <ul style="list-style-type: none"> Watcher: check the system status, I/O status, robot pose, and alarms Operator: Operate a robot based on the existing scripts without programming Programmer: Program, Teach Manager: Set parameters <p>Please select user mode based on site requirements</p>
9	<p>Emergency stop switch</p> <p>Press and hold it in an emergency, the drive power supply of robot motors will be powered off for emergency braking</p>
10	You can browse user guide and application case on welcome page

1.2 Settings

Before teaching or running robot programs, a series of settings are required, including motion parameter setting, language selecting, user mode selecting and process setting.

1.2.1 Setting Motion Parameter

You can set the velocity, acceleration or other parameters in different coordinate systems when jogging a robot or running robot programs. After setting the parameters, please click **Save**. Click

> **Config** > **RobotParams** to enter **RobotParams** interface.

- **Teach Joint Parameter:** Set the maximum velocity and acceleration in the Joint coordinate system when jogging a robot. The jogging parameters of a 6-axis robot in the Joint coordinate system are as shown in Figure 1.2.

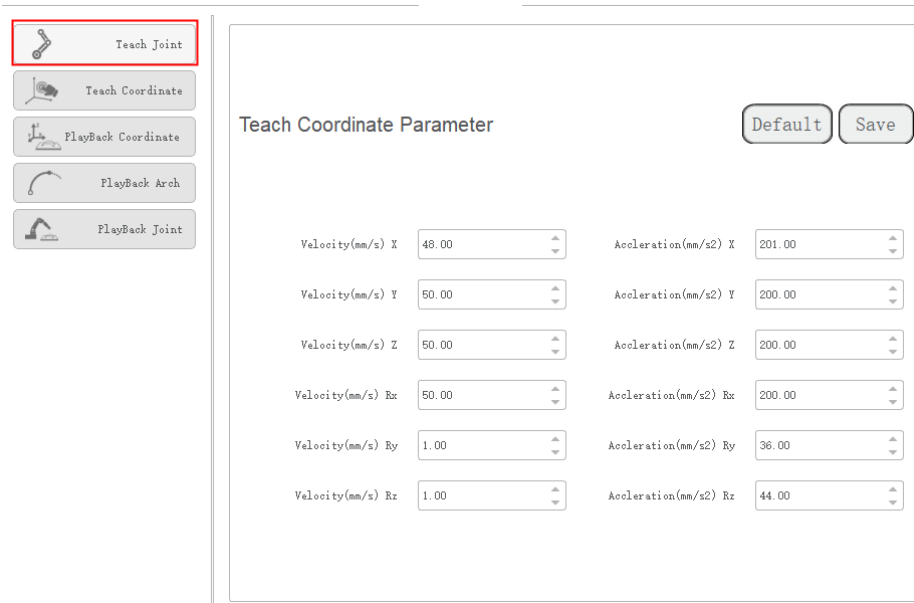


Figure 1.2 Jogging parameters in the Joint coordinate system

NOTE

If the robot is a SCARA type, the related parameters of J5 and J6 are invalid.

- Set the maximum velocity and acceleration in the Cartesian coordinate system when jogging a robot. The jogging parameters of a 6-axis robot in the Cartesian coordinate system are as shown in Figure 1.3.

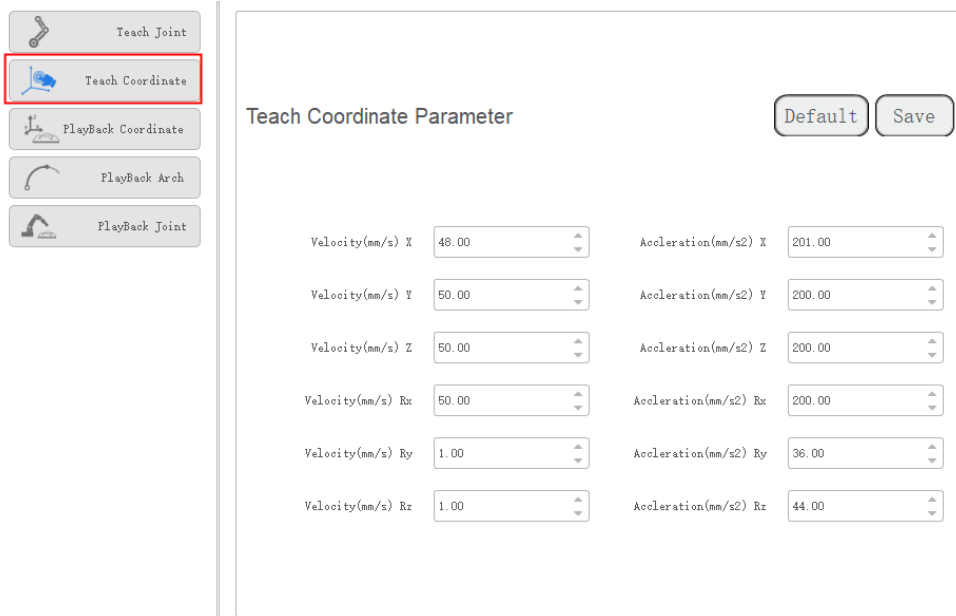


Figure 1.3 Jogging parameters in the Cartesian coordinate system

NOTE

If the robot is a SCARA type, **Rx** indicates the R-axis. The related parameters of **Ry** and **Rz** are invalid.

- **Playback Joint Parameter:** Set the maximum velocity, acceleration, and jerk in the Joint coordinate system when running robot programs. The playback parameters of a 6-axis robot in the Joint coordinate system are as shown in Figure 1.4.

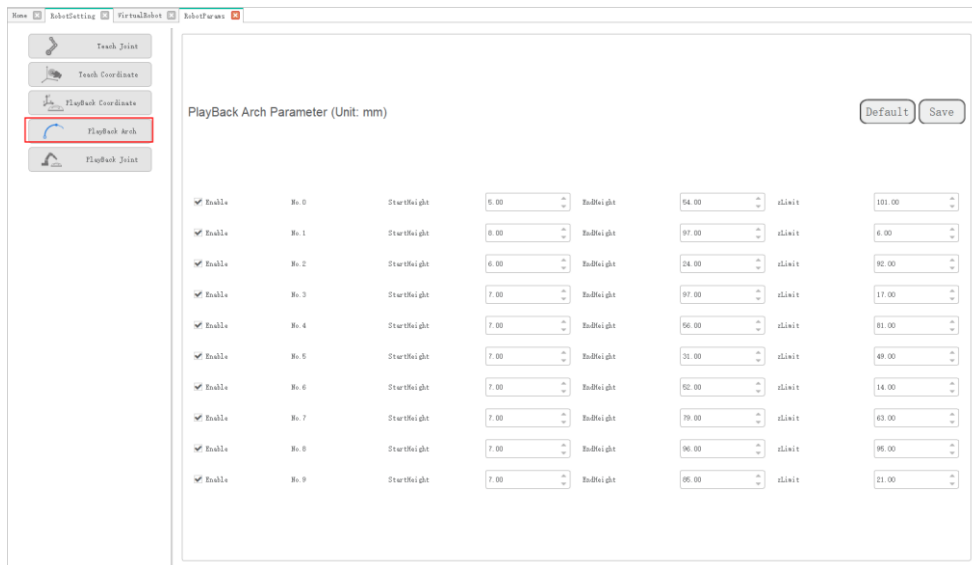


Figure 1.4 Playback parameters in the Joint coordinate system

NOTE

If the robot is a SCARA type, the related parameters of J5 and J6 are invalid.

- Playback Coordinate Parameter: Set the maximum velocity, acceleration and jerk in the Cartesian coordinate system when running robot programs. The playback parameters of a 6-axis robot in the Cartesian coordinate system are as shown in Figure 1.5.

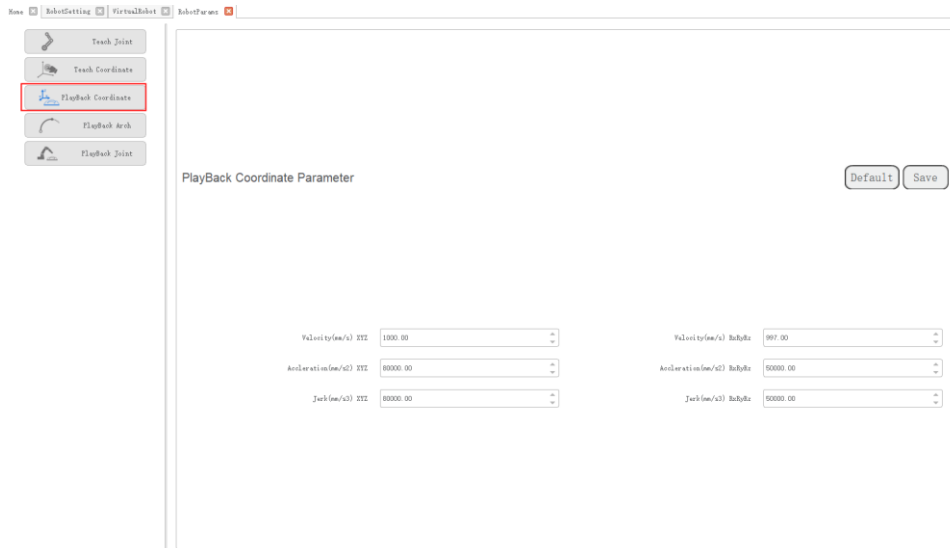


Figure 1.5 Playback parameters in the Cartesian coordinate system

NOTE

If the robot is a SCARA type, **RxRyRz** indicates the R-axis.

- Playback Arch Parameter: If the motion mode is **Jump** when running robot programs, you need to set **StartHeight**, **EndHeight**, and **zLimit**.

You can set 10 sets of Jump parameters. Please set and select any set of parameters for calling Jump command during programming, as shown in Figure 1.6.

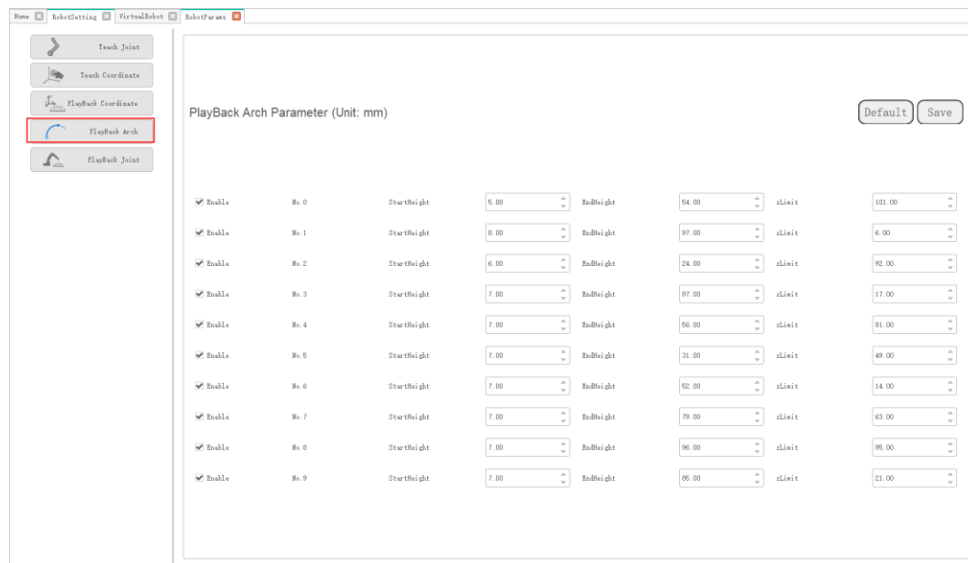


Figure 1.6 Jump parameters

1.2.2 Setting User Coordinate System

When the position of workpiece is changed or a robot program needs to be reused in multiple processing systems of the same type, you can create coordinate systems on the workpiece to simplify programming. There are totally 10 groups of User coordinate systems, of which the first one is defined as the Base coordinate system by default and cannot be changed. And the others can be customized by users.

NOTICE

When creating a User coordinate system, please make sure that the reference coordinate system is the Base coordinate system. Namely, the User coordinate system icon should be **User: 0** when creating a User coordinate system.

1.2.2.1 Setting User Coordinate System of SCARA Robot

User coordinate system of a SCARA robot is created by two-point calibration method: Move the robot to two points **A(x1, y1, z1)** and **B(x2, y2, z2)**. Point A is defined as the origin and the line from point A to point B is defined as the positive direction of X-axis. And then the Y-axis and Z-axis can be defined based on the right-handed rule, as shown in Figure 1.7.

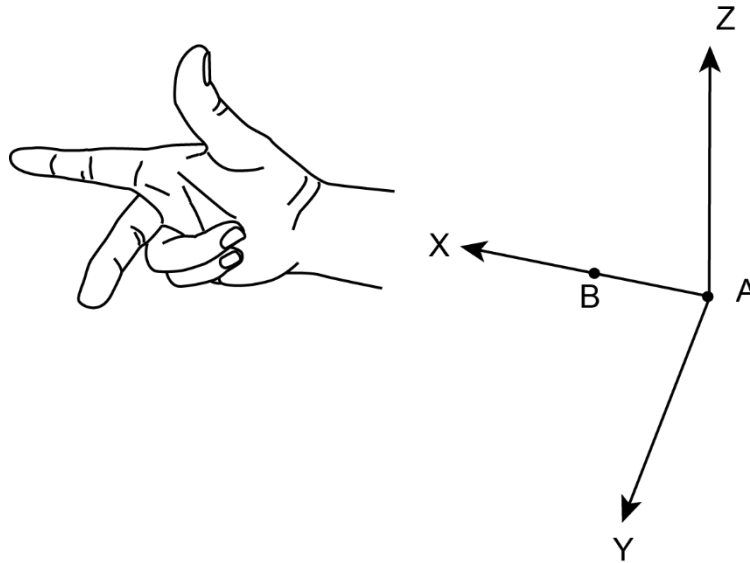


Figure 1.7 Two-point calibration

Take the establishment of User 1 coordinate system as an example.

Prerequisites

- The robot has been powered on.
- The DobotSCStudio has been in the manual mode.

Procedure

Step 1 Click  > **Config** > **GlobalCoordinate** > **Coordinate User**.

The **Coordinate User** page is displayed, as shown in Figure 1.8.

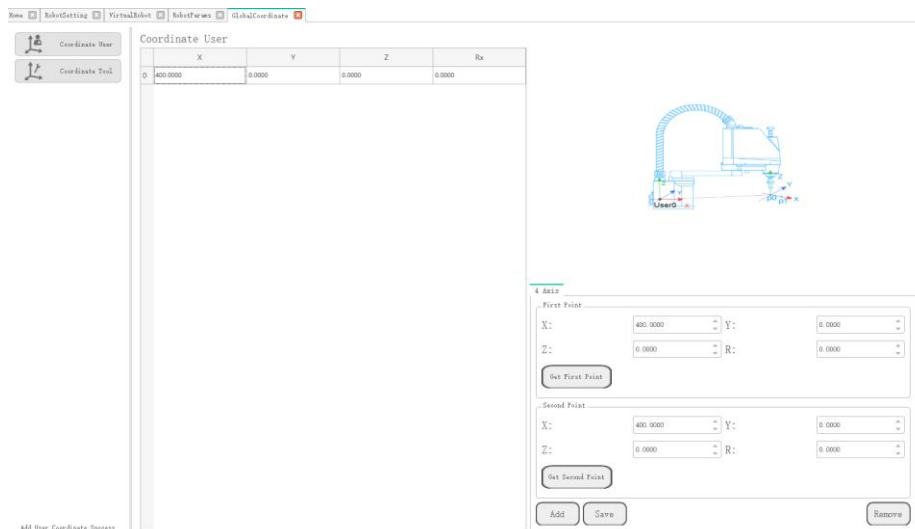


Figure 1.8 User coordinate system page

Step 2 Enable the motor and jog the robot to a point, then click **Get First Point** on the **First**

Point section to obtain the coordinates of the first point.

Step 3 Enable the motor and jog the robot to another point, then click **Get Second Point** on the **Second Point** section to obtain the coordinates of the second point.

Step 4 Click **Add** to generate the User 1 coordinate system.

Step 5 Select **User: 1** on Jog interface.

You can use the **User 1** coordinate system for teaching and programming.

1.2.2.2 Setting User Coordinate System of 6-axis Robot

User coordinate system of a 6-axis robot is created by three-point calibration method: Move the robot to three points $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$, and $C(x_3, y_3, z_3)$. Point A is defined as the origin and the line from point A to Point B is defined as the positive direction of X-axis. The line that point C is perpendicular to X-axis is defined as the position direction of Y-axis. And then the Z-axis can be defined based on the right-handed rule, as shown in Figure 1.9.

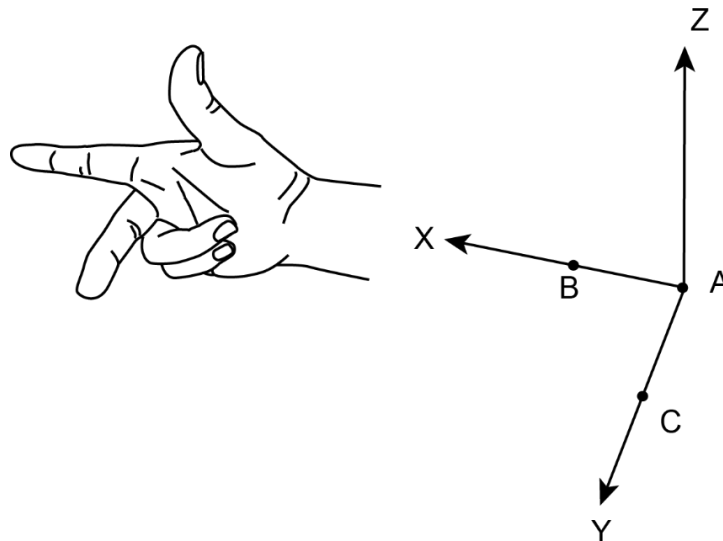


Figure 1.9 Three-point calibration

Take the establishment of User 1 coordinate system as an example.

Prerequisites

- The robot has been powered on.
- The DobotSCStudio has been in the manual mode.

Procedure

Step 1 Click  > **Config** > **GlobalCoordinate** > **Coordinate User6Axis**.

The **Coordinate User** page is displayed, as shown in Figure 1.10.

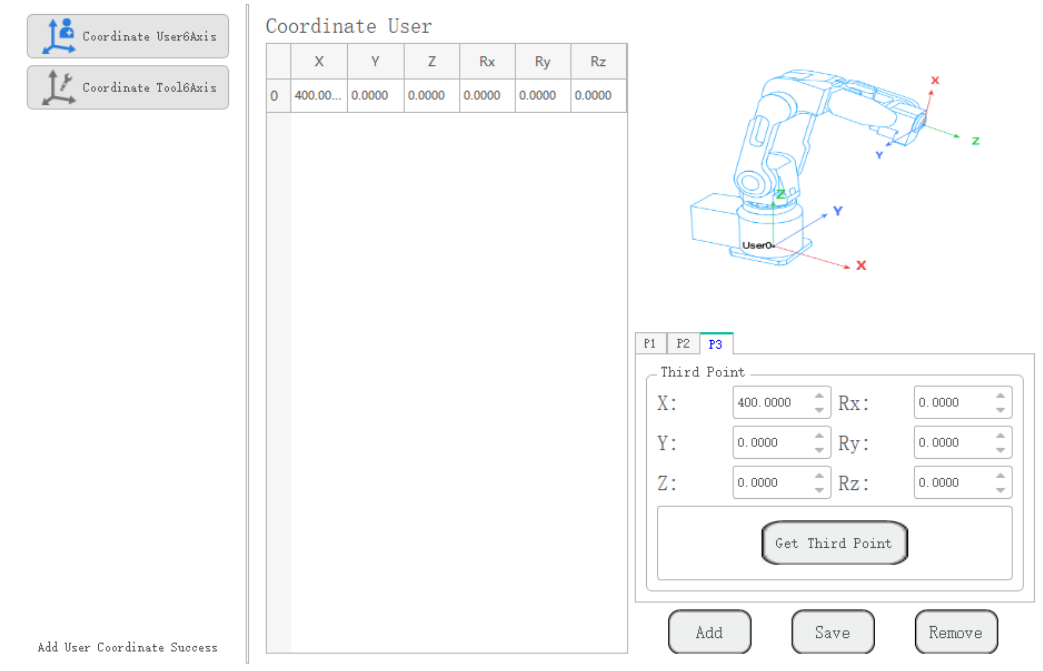


Figure 1.10 User coordinate system page

NOTE

Rx, Ry, Rz are the orientation data, which are designated by rotating the tool center point (TCP) around the X, Y, Z axes under the selected User coordinate system.

- Step 2** Enable the motor and jog the robot to the first point, then click **Get First Point** on the **P1** tab to obtain the coordinates of the first point.
- Step 3** Enable the motor and jog the robot to the second point, then click **Get Second Point** on the **P2** tab to obtain the coordinates of the second point.
- Step 4** Enable the motor and jog the robot to the third point, then click **Get Third Point** on the **P3** tab to obtain the coordinates of the third point.
- Step 5** Click **Add** and **Save** to generate the User 1 coordinate system.
- Step 6** Select **User: 1** on Jog interface.

You can use the User 1 coordinate system for teaching and programming.

1.2.3 Setting Tool Coordinate System

When an end effector such as welding gun, gripper is mounted on the robot, the Tool coordinate system is required for programming and operating a robot. For example, you can use multiple grippers to carry multiple workpieces simultaneously to improve the efficiency by setting each gripper to a Tool coordinate system.

There are totally 10 groups of Tool coordinate systems. Tool 0 coordinate system is the predefined Tool coordinate system which is located at the robot flange and cannot be changed.

⚠ NOTICE

When creating a Tool coordinate system, please make sure that the reference coordinate system is the predefined Tool coordinate system. Namely, the Tool coordinate system icon should be `Tool: 0` when creating a Tool coordinate system.

1.2.3.1 Setting Tool Coordinate System of SCARA Robot

Tool coordinate system of SCARA robot is created by two-point calibration method: After an end effector is mounted, please adjust the direction of this end effector to make the TCP (Tool Center Point) align with the same point (reference point) in two different directions, for obtaining the position offset to generate a Tool coordinate system, as shown in Figure 1.11.

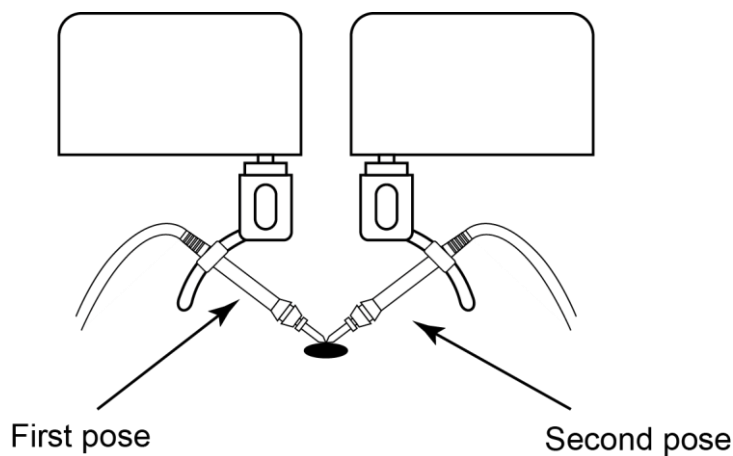


Figure 1.11 Two-point calibration method

Take the establishment of Tool 1 coordinate system as an example.

Prerequisites

- The robot has been powered on.
- The DobotSCStudio has been in the manual mode.

Procedure

Step 1 Mount an end effector on the robot. The detailed instructions are not described in this topic.

Step 2 Click  > **Config** > **GlobalCoordinate** > **Coordinate Tool**.

The Coordinate Tool page is displayed, as shown in Figure 1.12.

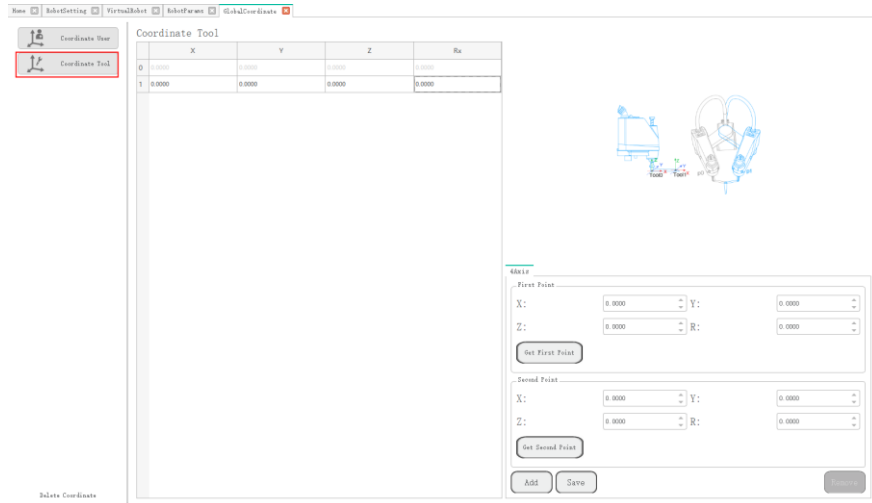


Figure 1.12 Tool coordinate page

- Step 3** Enable the motor and jog the robot to the reference point in the first direction, then click **Get First Point** on the **First Point** section to obtain the coordinates of the first point.
- Step 4** Enable the motor and jog the J4-axis, then jog other axes to the reference point in the second direction, and then click **Get second Point** on the **second Point** section to obtain the coordinates of the second point.
- Step 5** Click **Add** to generate the Tool 1 coordinate system.
- Step 6** Select **Tool: 1** on Jog interface.
You can use the Tool 1 coordinate system for teaching and programming.

Result Validation

Make the TCP align with the fixed point and then jog the R-axis. If the robot can rotate around this point, it indicates that the Tool coordinate system is created successfully.

1.2.3.2 Setting Tool Coordinate System of 6-axis Robot

Tool coordinate system of a 6-axis robot is created by three-point calibration method (TCP +ZX): After the end effector is mounted, please adjust the direction of the end effector, to make TCP (Tool Center Point) align with the same point (reference point) in three different directions, for obtaining the position offset of the end effector, and then jog the robot to the other three points (A, B, C) for obtaining the angle offset, as shown in Figure 1.13.

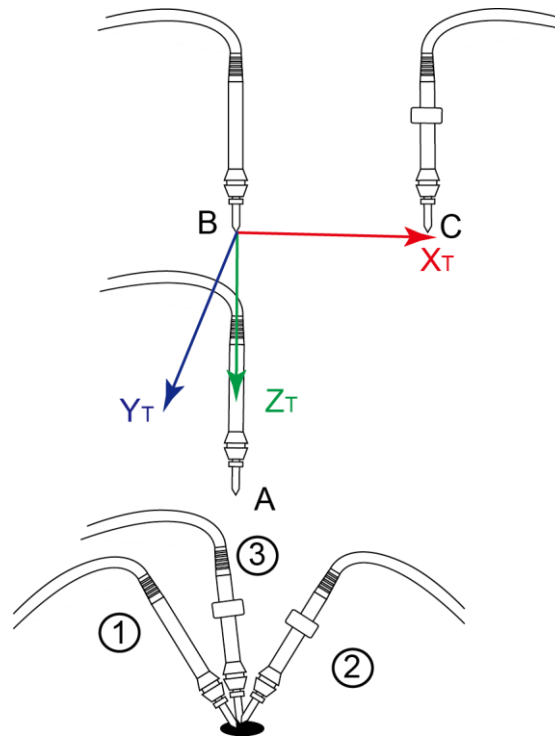


Figure 1.13 Three points calibration method (TCP+ZX)

Take the establishment of Tool 1 coordinate system as an example.

Prerequisites

- The robot has been powered on.
- The DobotSCStudio has been in the manual mode.

Procedure

Step 1 Mount an end effector on the robot. The detailed instructions are not described in this topic.

Step 2 Click  > **Config** > **GlobalCoordinate** > **Coordinate Tool6Axis**.

The Coordinate Tool page is displayed, as shown in Figure 1.14.

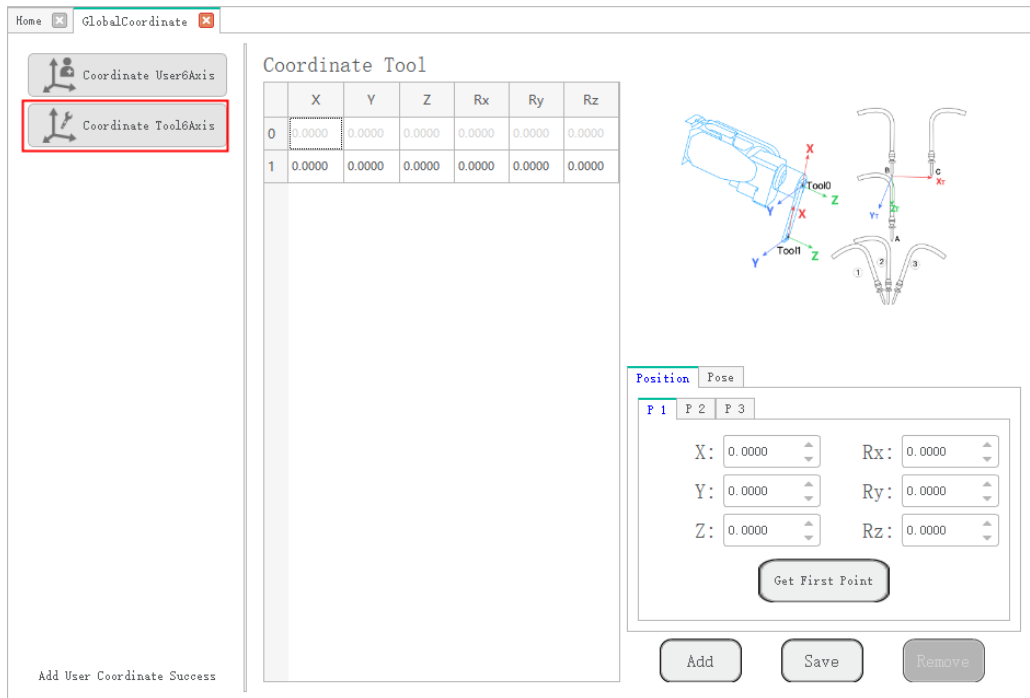


Figure 1.14 Tool Coordinate page

NOTE

Rx, Ry, Rz are the orientation data, which are designated by rotating the tool center point (TCP) around the X, Y, Z axes under the selected Tool coordinate system.

- Step 3** Enable the motor and jog the robot to the reference point in the first direction, then click **Get First Point** on the **P1** tab of the **Position** page to obtain the coordinates of the first point.
- Step 4** Enable the motor and jog the robot to the reference point in the second direction, then click **Get Second Point** on the **P2** tab of the **Position** page to obtain the coordinates of the second point.
- Step 5** Enable the motor and jog the robot to the reference point in the third direction, then click **Get Third Point** on the **P3** tab of the **Position** page to obtain the coordinates of the third point.
- Step 6** Enable the motor and jog the robot to the reference point (point **A**) in the vertical direction, then click **Get First Point** on the **P1** tab of the **Pose** page to obtain the fourth point.
- Step 7** Enable the motor and jog the Z-axis to a point (point **B**) along the positive direction, then click **Get Second Point** on the **P2** tab of the **Pose** page to obtain the fifth point. This step defines the Z-axis.
- Step 8** Enable the motor and jog the X-axis to another point (point **C**), then click **Get Third Point** on the **P3** tab of the **Pose** page to obtain the sixth point.

The three points (A,B,C) cannot lie in the same line.

This step defines the X-axis, and the Y-axis can be defined based on the right-handed rule.

Step 9 Click **Add** to generate the Tool 1 coordinate system.

Step 10 Click **Select User: 1** on Jog interface.

You can use the Tool 1 coordinate system for teaching and programming.

1.2.4 Homing

After some parts (motors, reduction gear units) of the robot have been replaced or the robot has been hit, the origin of the robot will be changed. You need to reset the origin.

1.2.4.1 Homing of SCARA Robot

Prerequisites

The robot has been powered on.

Procedure

Step 1 Make the robot motor in the disabled status, and put the robot in the original position, as shown in Figure 1.15.

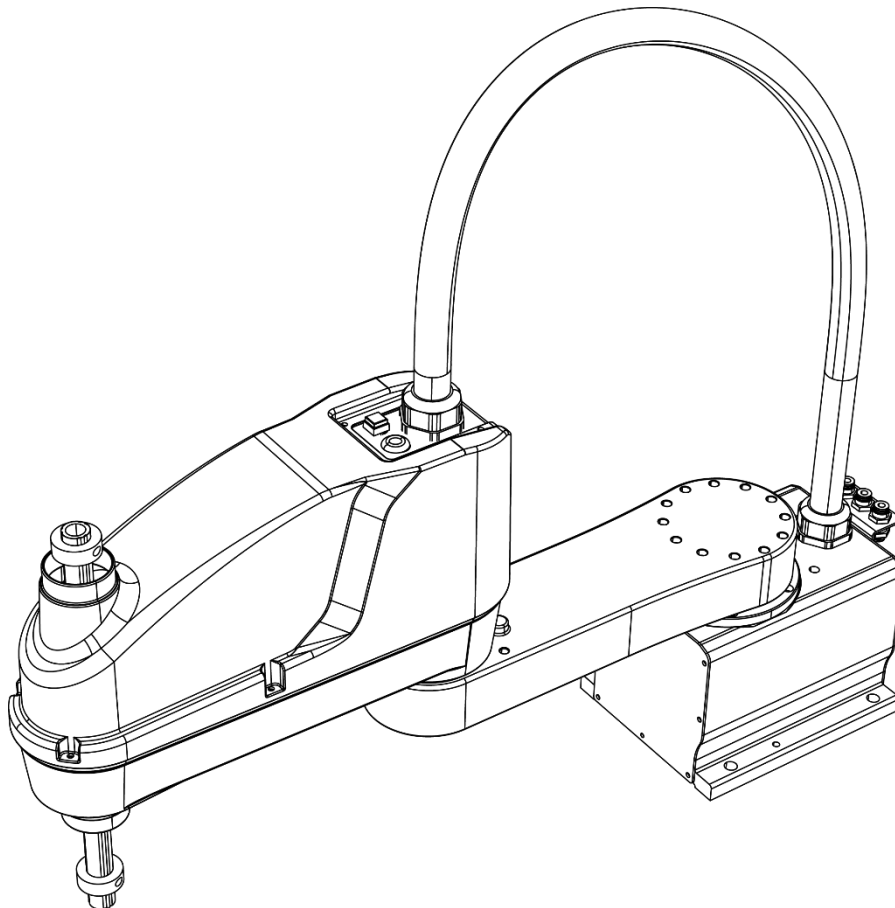


Figure 1.15 Original position

NOTE

There is a keyway on each joint. When moving the robot, the position where the keyways of the adjacent joints are aligned is called the original position of the corresponding joint, as shown in Figure 1.16.

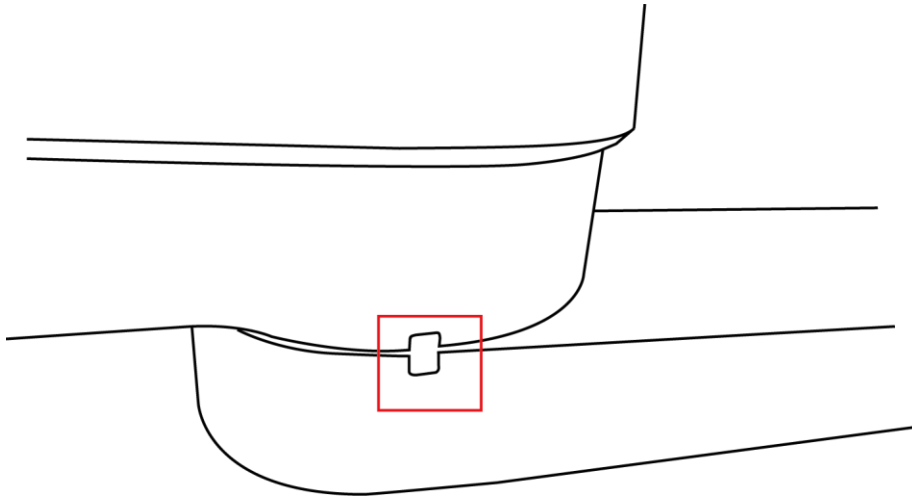


Figure 1.16 Keyway position

- Step 2** Rotate the mechanical stop clockwise on the ball screw to the limited position and then rotate it counter-clockwise about 180° or 360° . Figure 1.17 shows a mechanical stop.

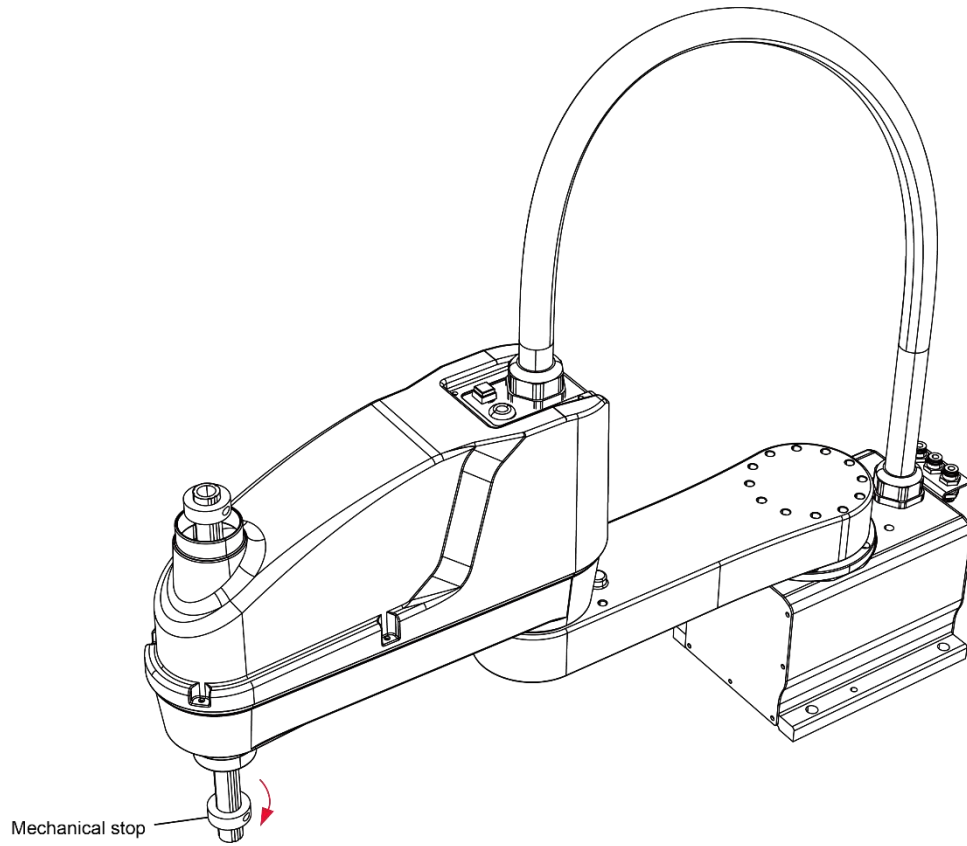


Figure 1.17 Mechanical stop

 NOTE

Mechanical stop is used to prevent the robot from running out of the workspace, to avoid damage to the robot and operators.

Step 3 Make DobotSCStudio in the manual mode and enable the robot motor.

Step 4 Click  > **Config** > **Robotsetting** > **Home**.

The home page is displayed, as shown in Figure 1.18.

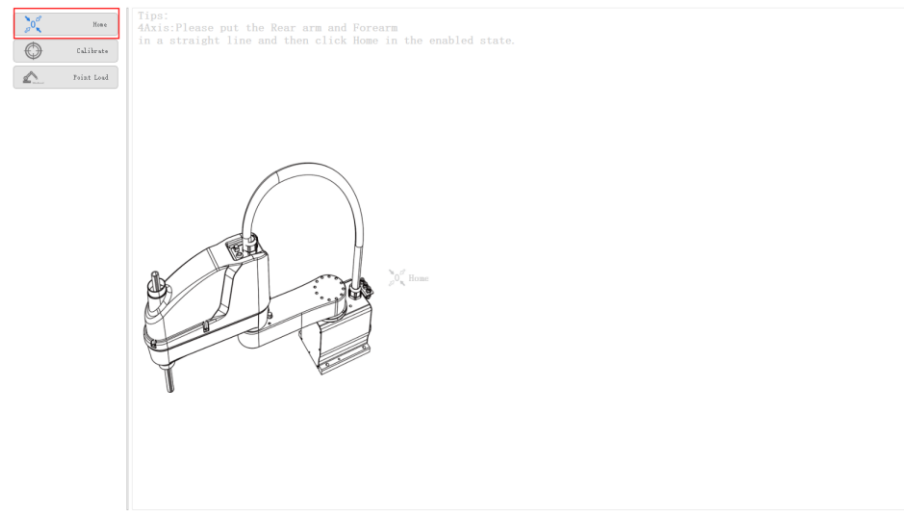



Figure 1.18 Homing page

Step 5 Click  on the homing page.

If the homing procedure is successful, **Home succeeded!** is displayed on the message window.

During the homing procedure, the robot will not move and will set the current position as the homing point. The homing position is shown in Figure 1.19.

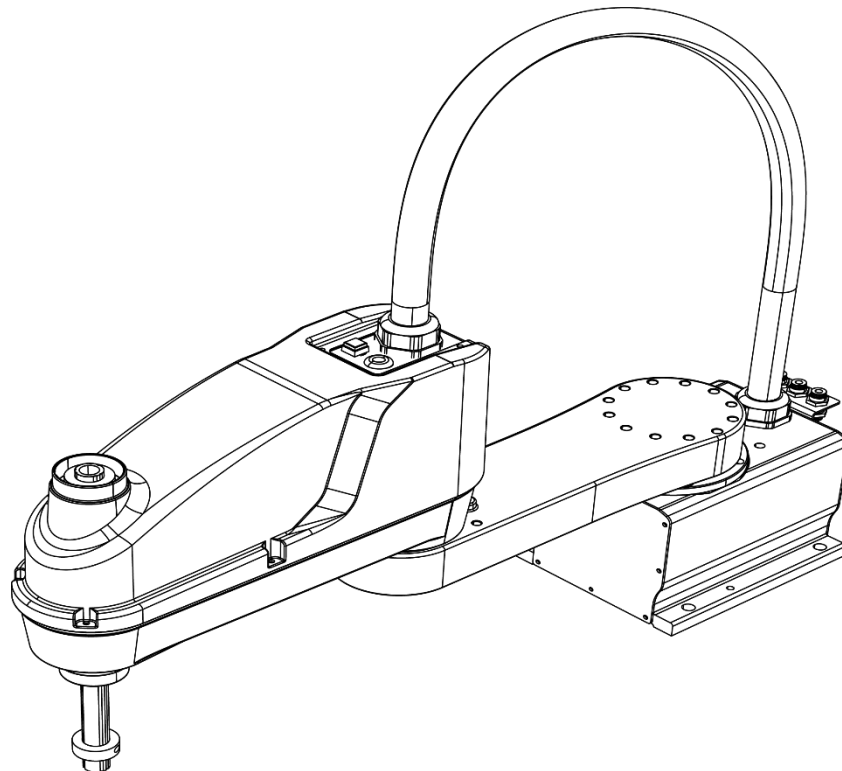


Figure 1.19 Homing position

1.2.4.2 Homing of 6-axis Robot

Prerequisites

The robot has been powered on.

Procedure

- Step 1** Make the robot motor in the disabled status and put the robot in the original position, as shown in Figure 1.20.

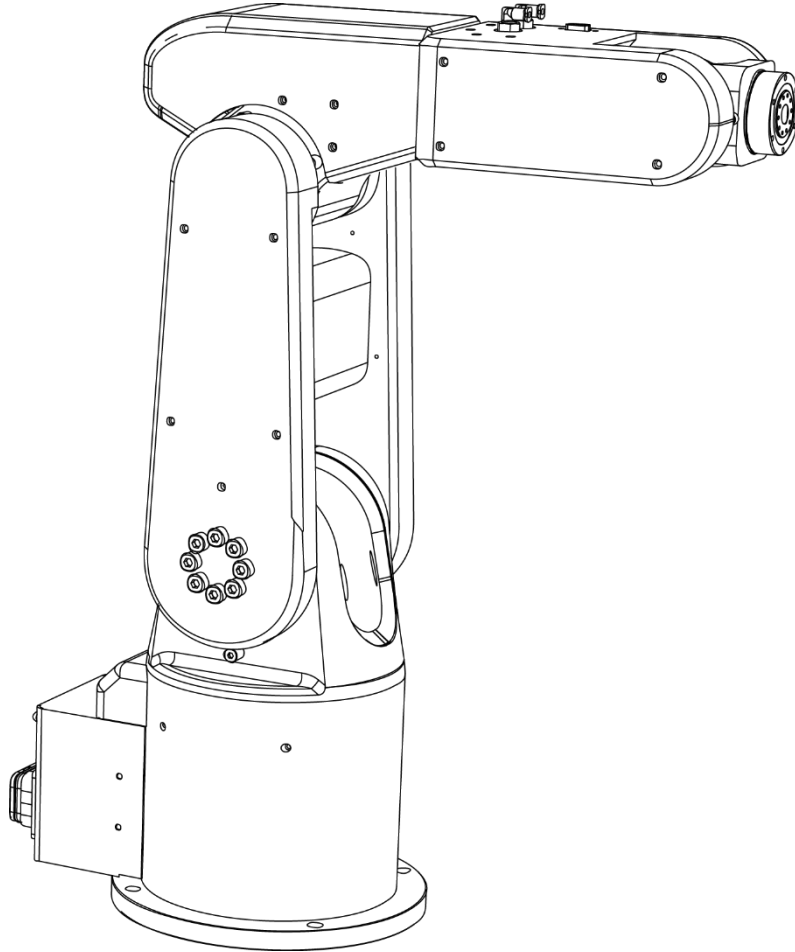


Figure 1.20 Original position

NOTE

There is a keyway on each joint. When moving the robot, the position where the keyways of the adjacent joints are aligned is called the original position of the corresponding joint, as shown in Figure 1.21.

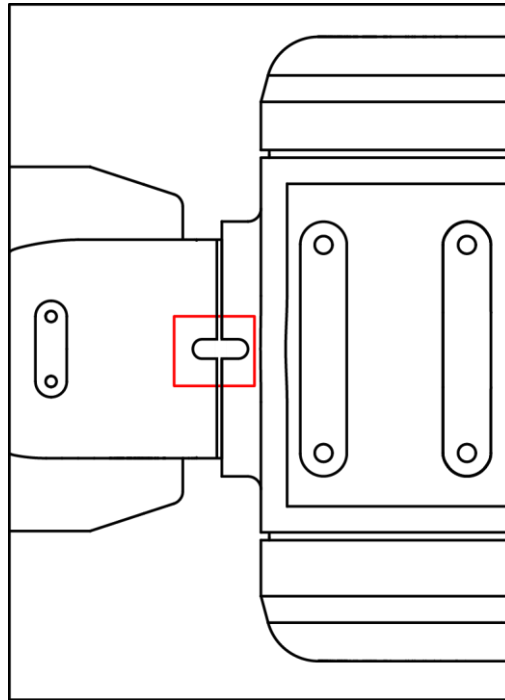


Figure 1.21 Keyway position

Step 2 Make DobotSCStudio in the manual mode and enable the robot motor.

Step 3 Click  > **Config** > **Robotsetting** > **Home**.

The homing **page** is displayed, as shown in Figure 1.22.

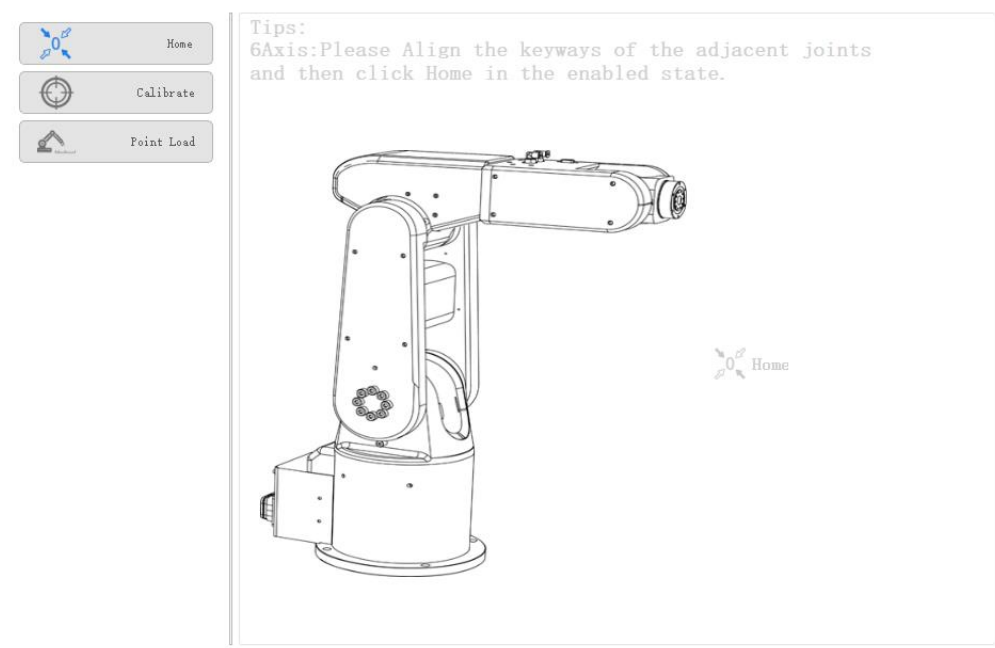



Figure 1.22 Homing page

Step 4 Click  on the homing page.

If the homing procedure is successful, **Home succeeded!** is displayed on the message window.

During the homing procedure, the robot will not move and will set the current position as the homing point. The homing position is shown in Figure 1.23.

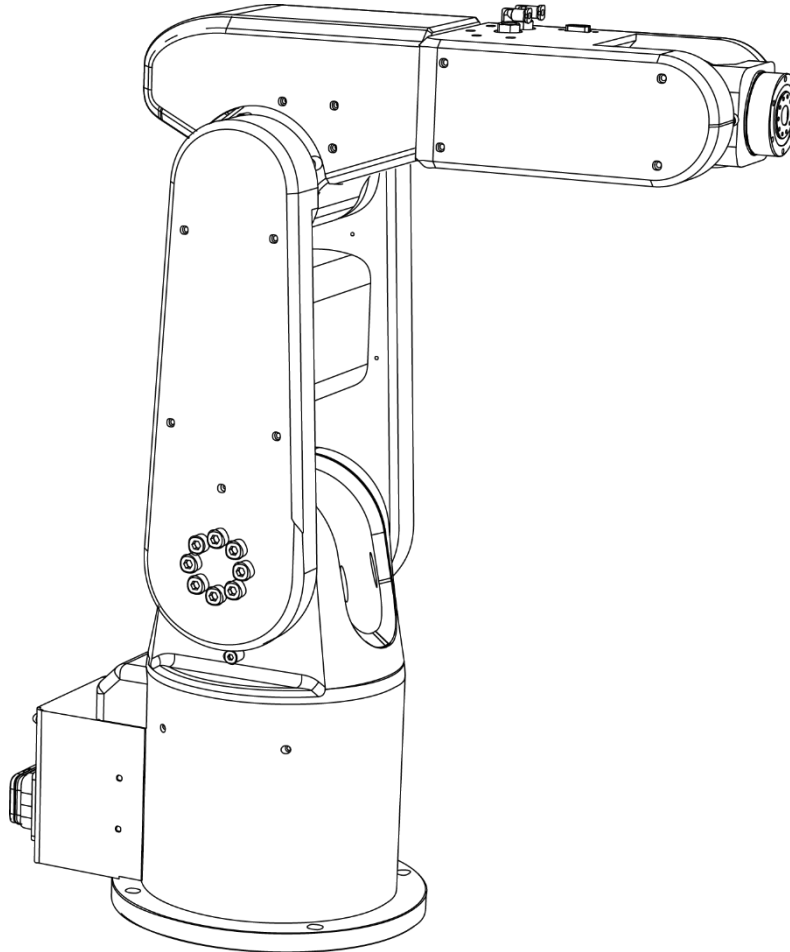


Figure 1.23 Homing position

1.2.5 Calibration

Before being shipped out, the robot has been calibrated. You need to re-calibrate it if higher absolute precision is required in real applications.

For SCARA robot: Generally, the robot moves to the same point with different arm orientations, the J2 coordinates are axisymmetric. If not, absolute precision will be decreased. It is necessary to make the J2 coordinates axisymmetric by compensating the joint angle of J2 to improve absolute precision.

This topic takes a SCARA robot as an example to describe how to calibrate.


Prerequisites

- The robot has been powered on.

- The calibration kit has been mounted.

Procedure

Step 1 Switch DobotSCStudio to the manual mode.

Step 2 Click  > **Config** > **Robotsetting** > **Calibrate**.

The calibration page is displayed, as shown in Figure 1.24.



Figure 1.24 Calibration page

Step 3 Enable the robot motor.

The following steps must be performed in the enabled status.

Step 4 Jog the robot with lefty hand orientation to a point on the calibration plate.

Step 5 Click **Get P1** on the **Left** section.

The value displayed on the **Left** section is J2 coordinate with lefty hand orientation.

Step 6 Jog the robot to lift a certain height and then jog to the same point in **Step 4** with righty hand orientation, as shown in Figure 1.25.

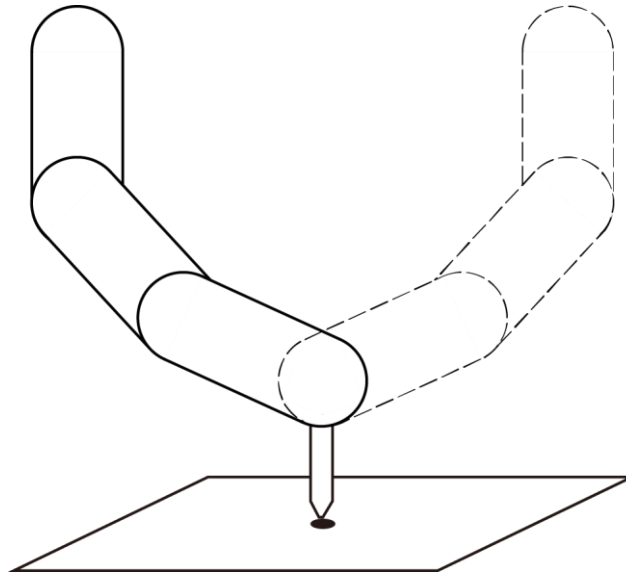


Figure 1.25 Calibration

Step 7 Click **Get P2** on the **Right** section.

The value displayed on the **Right** section is J2 coordinate with righty hand orientation.

Step 8 Click **Calibrate**.

If the calibration is successful, The **Calibration Succeeded!** is displayed on the message window.

1.2.6 VirtualRobot

When user jogs or runs robot, the virtual simulation interface can be used to view the robot movement in real time.

1.2.7 Log

You can understand the historical operation of the robot by viewing the log. The log can be screened according to three types of logs: user operation, control error and servo error. Click **Reset** to clear the log.




Figure 1.26 Log

1.2.8 Language

Click **Config> BasicConfig> Language** enter the language switching interface, you can switch to Chinese or English.

1.2.9 Network Service

The robot system can be communicated with external equipment by the Ethernet interface which supports TCP, UDP and Modbus protocols. The default IP address is **192.16.5.1**. In real applications, if the TCP or UDP protocol is used, the robot system can be a client or a server based on site requirements; if the Modbus protocol is used, the robot system only can be the Modbus slave, and the external equipment is the master.

You can modify the IP address on the  **>Config> NetworkSetting** page, as shown in Figure 1.27. The IP address of the robot system must be in the same network segment of the external equipment without conflict.

Auto IP Address:
 Manual IP Address:

Manual

IP Address:	192	.	168	.	5	.	1
subnet mask:	255	.	255	.	255	.	0
default gateway:	192	.	168	.	5	.	1

Figure 1.27 IP address setting

Figure 1.28 and Figure 1.29 show the connections between the robot system and external equipment.

- If the robot system connects to the external equipment directly or with a switchboard, please select **Manual IP Address** and modify **IP Address**, **subnet mask**, **default gateway**, and then click **Save**.
- If the robot system connects to the external equipment with a router, please select **Auto IP Address** to assign IP address automatically, and then click **Save**.

 NOTICE

Please DO NOT insert the network cable into the WAN interface when using a router for the connection.

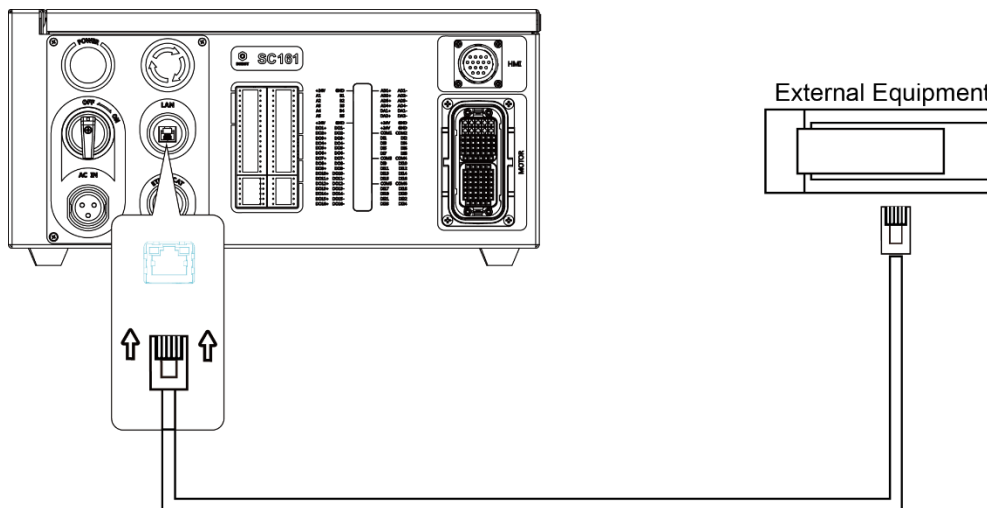


Figure 1.28 Connect robot system to external equipment directly

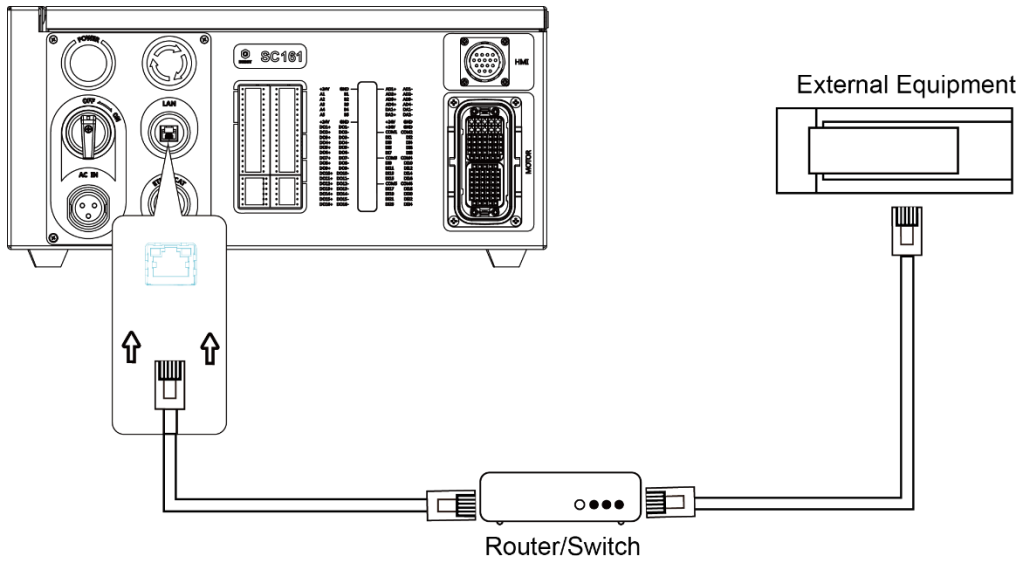



Figure 1.29 Connect robot system to external equipment with router or switchboard

1.3 Monitor

1.3.1 I/O Monitor

Click  > **Config** > **IOMonitor**, Figure 1.30 shows the I/O monitor.

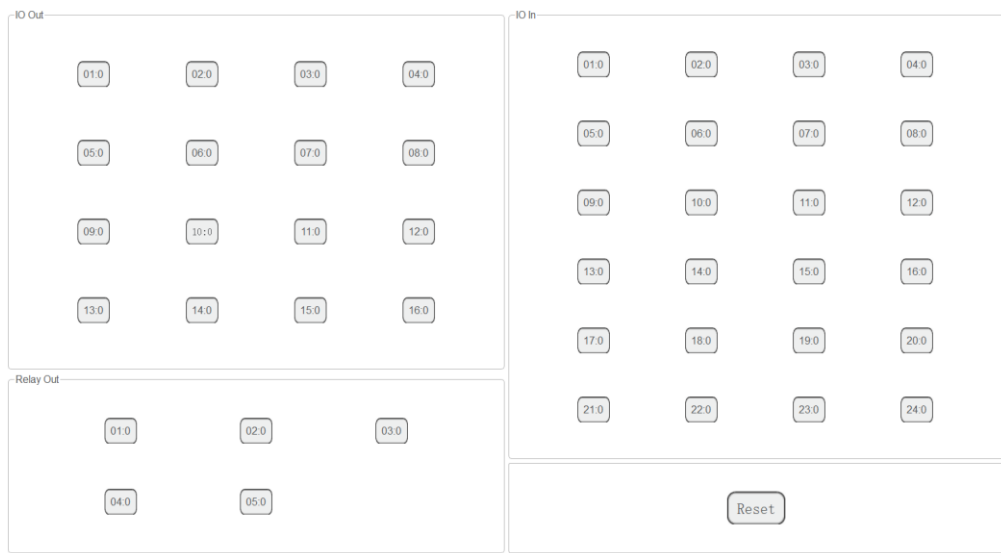


Figure 1.30 I/O monitor page

There are two features: Output and monitor

- Output: Set the output status in the manual mode.
- Monitor: Check the status of the input and output. In the auto mode, the status of the output and input cannot be modified.

1.4 Remote Control

External equipment can send commands to a robot by different remote control modes, such as remote I/O mode and remote Modbus mode. The default mode is Teaching mode when the robot is shipped out. When you need to set the remote mode, please set it on DobotSCStudio with the robot motor in the disabled state.

NOTICE

- Robot rebooting is not required when switching the remote mode.
- The emergency stop switch on the hardware is always available no matter what mode the robot system is in.
- Please **DO NOT** switch the remote mode when the robot is running in the current remote mode. You need to exit the current mode and then switch to the other remote mode. Namely, please stop the robot running and then switch the mode.
- If the robot motor is in the enabled status, the remote control cannot be used. Otherwise, an alarm will be triggered. Please activate the remote control in the disabled status.

1.4.1 Remote I/O

When the remote mode is I/O mode, external equipment can control a robot in this mode. The specific I/O interface descriptions are shown in Table 1.2.

Table 1.2 Specific I/O interface description

I/O interface	Description
Input (For external control)	
DI 11	Clear alarm
DI 12	Continue to run
DI 13	Pause running in the I/O mode
DI 14	Stop running and exit the I/O mode
DI 15	Start to run in the I/O mode
DI 16	Emergency stop and exit the I/O mode
Output (For displaying the status)	
DO 13	Ready status
DO 14	Pause status
DO 15	Alarm status
DO 16	Running status

NOTICE

All input signals are rising-edge triggered.

Prerequisites

- The project to be running in the remote mode has been prepared.
- The external equipment has been connected to the robot system by the I/O interface. The specific I/O interface description is shown in Table 1.2.
- The robot has been powered on.

NOTE

The details on how to connect external equipment and use it are not described in this topic.

Procedure

Step 1 Click > **Config>Offline**.

The remote control page is displayed, as shown in Figure 1.31.

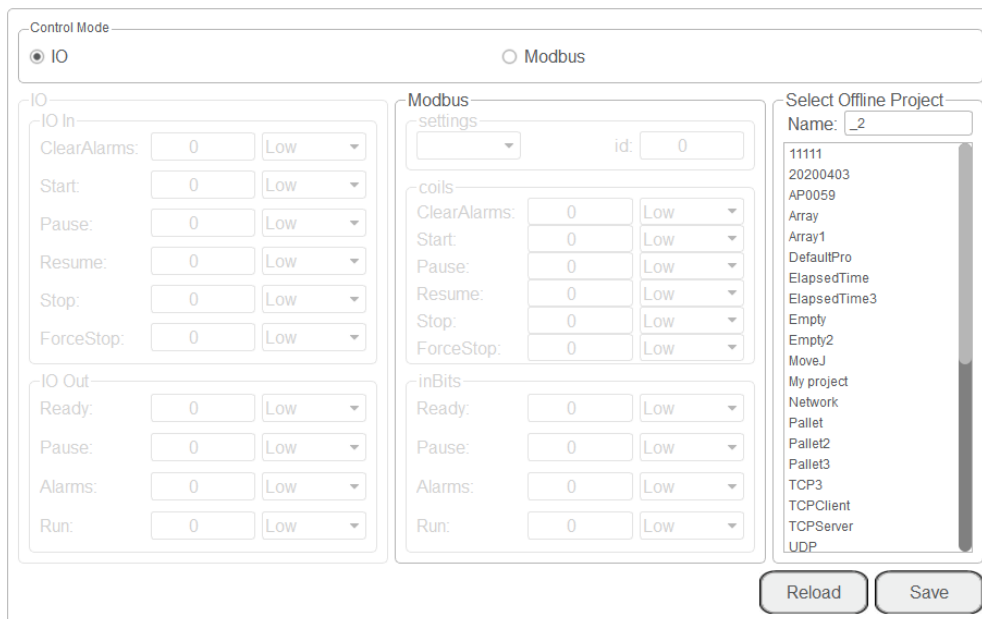


Figure 1.31 Remote control page

Step 2 Select **IO** on the **Control Mode** section and select the offline project on the **Select Offline Project** section.

The **Save success, now remote control mode is IO** page is displayed.

Right now, only the emergency stop button is available.

Step 3 Trigger the starting signal on the external equipment.

The robot will move as the selected offline project. If the stop signal is triggered, the remote I/O mode will be invalid.

1.4.2 Remote Modbus

When the remote mode is Modbus mode, external equipment can control a robot in this mode. For details about Modbus registers, please see *2.15.1 Modbus Register Description*. The specific Modbus register descriptions are shown in Table 1.3.

Table 1.3 Specific Modbus register description

Register address (Take a PLC as an example)	Register address (Robot system)	Description
Coil register		
00001	0	Start running in the remote Modbus mode
00002	1	Pause running in the remote Modbus mode
00003	2	Continue to run
00004	3	Stop to run and exit the remote Modbus mode
00005	4	Emergency stop and exit the remote Modbus mode
00006	5	Clear alarm
Discrete input register		
10001	0	Auto-exit
10002	1	Ready status
10003	2	Pause status
10004	3	Running status
10005	4	Alarm status

Prerequisites

- The project to be running in the remote mode has been prepared.
- The robot has been connected to the external equipment with the Ethernet interface. You can connect them directly or via a router, please select based on site requirements.

The IP address of the robot system must be in the same network segment of the external equipment without conflict. You can modify the IP address on the **Config> NetworkSetting** page; the default port is **502** and cannot be modified.

- The robot has been powered on.

 NOTE

The details on how to connect external equipment and use it are not described in this topic.

Procedure

Step 1 Click  > **Config>Offline**.

The remote control page is displayed, as shown in Figure 1.32.

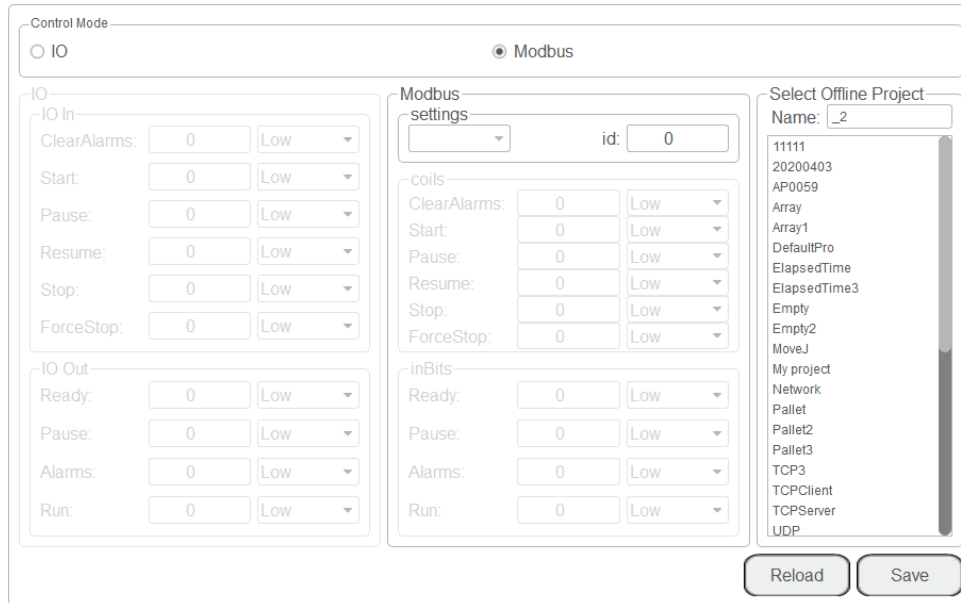


Figure 1.32 Remote control page

Step 2 Select **Modbus** on the **Control Mode** section and select the offline project on the **Select Offline Project** section.

The **Save success, now remote control mode is Modbus** page is displayed.

Right now, only the emergency stop button is available.

Step 3 Trigger the starting signal on the external equipment.

The robot will move as the selected offline project. If the stop signal is triggered, the remote Modbus mode will be invalid.

1.5 Programming

1.5.1 Project Description

The robot program is managed in project form, including teaching points list, global variables, and program files. Figure 1.33 shows the project structure.

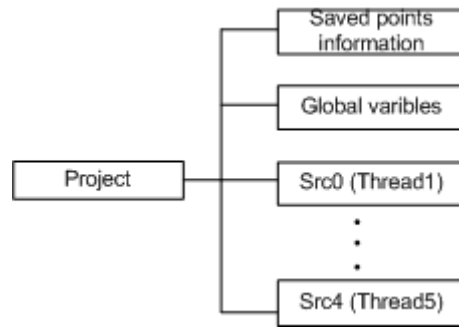


Figure 1.33 Project structure

1.5.2 Programming Interface Description

When programming a robot, please switch DobotSCStudio to the manual mode. Figure 1.34 shows the programming panel and Table 1.4 lists its description.

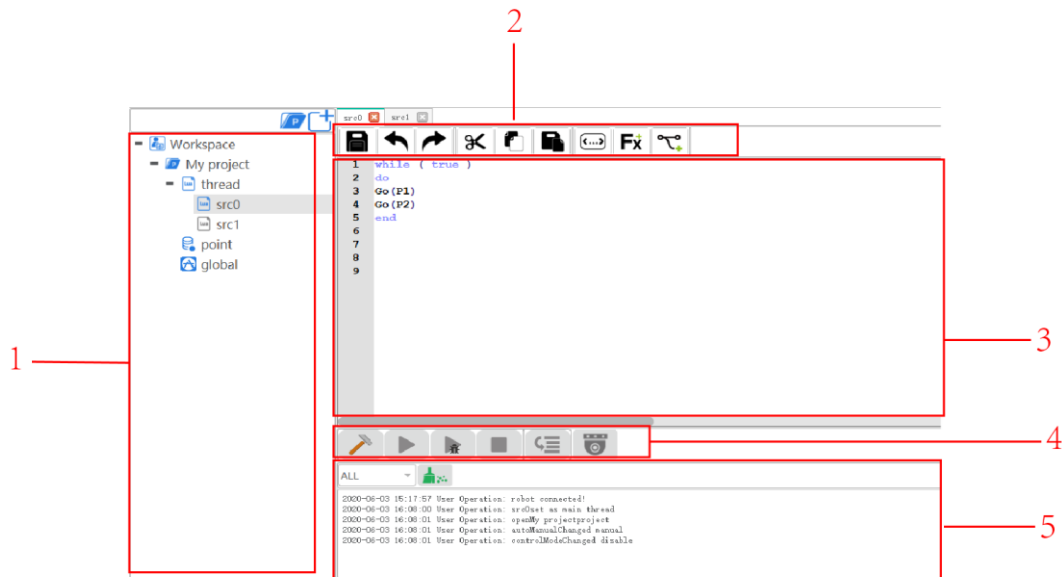


Figure 1.34 Programming panel










Table 1.4 Programming panel description

No.	Description
1	Project files <ul style="list-style-type: none"> • TeachPoint: Teach points. For details, please see <i>1.5.3.2 Teaching points</i> • Global: Define and initialize global variables or functions • Src0~Src4: Multithreaded files. The number of

No.	Description
	threads is related to CPU that is set when creating a project. Up to 5 threads can be executed simultaneously
2	Common buttons. For details, please see Table 1.5
3	Programming area
4	Running button, for details, please see Table 1.6
5	Debug result

Table 1.5 lists the common button description.

Table 1.5 Common button description

Icon	Description
	Save the project
	Cancel
	Redo
	Copy the selected codes
	Cut the selected codes
	Paste the selected codes
	Motion command libraries. For details, please see 2 <i>Program Language</i>
	Code comment
	Common operation instructions and process control instructions, for details, please see 2 <i>Program Language</i>

1.5.3 Programming Description

Take a SCARA robot as an example to describe how to program. Figure 1.35 shows the programming process.

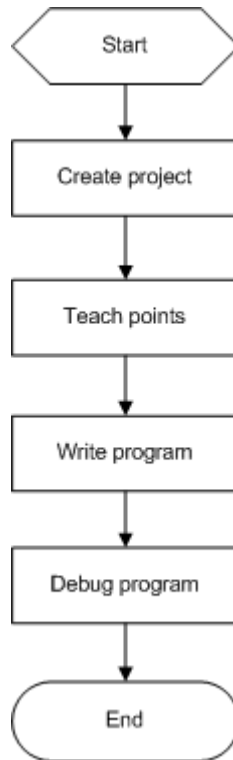


Figure 1.35 Programming process

1.5.3.1 Creating Project

Prerequisites

- The robot has been powered on.
- DobotSCStudio has been in the manual mode.

Procedure

Step 1 Click  .

The programming page is displayed, as shown in Figure 1.36.

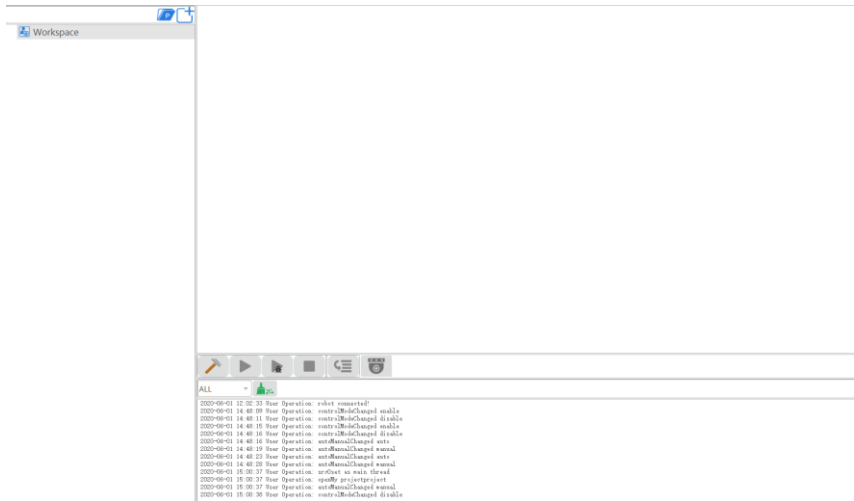



Figure 1.36 Programming page

Step 2 Click  enter project creating page, enter the project name, you can also select a template. Click **OK**.

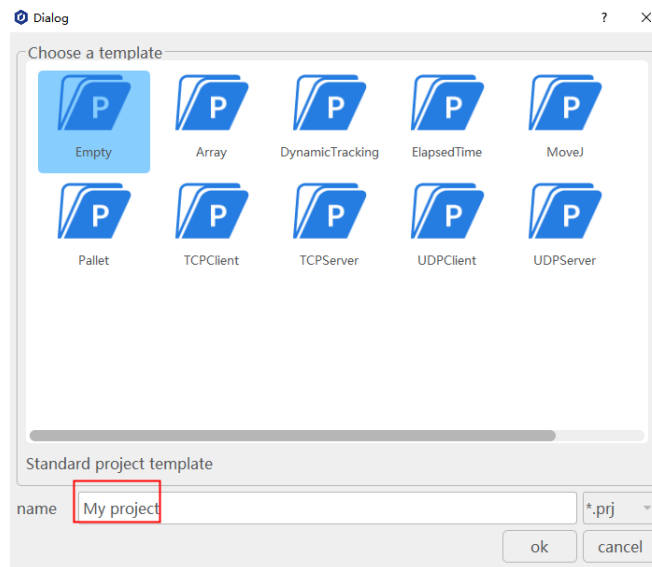


Figure 1.37 Create a project

Step 3 Set the number of threads based on site requirements, as shown in Figure 1.38. Right-click **thread** and click **New thread file**.

The maximum number of threads is **5**.

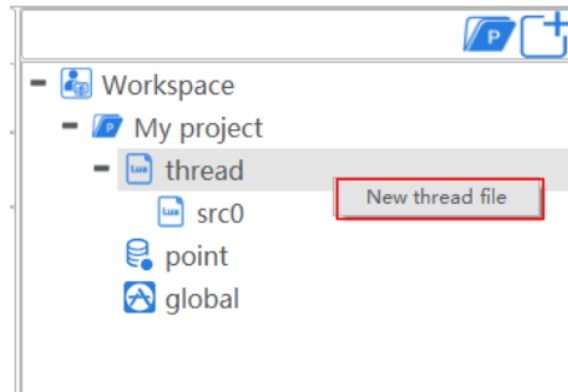


Figure 1.38 Create a project

Step 4 (Optional) Import the existing taught positions list.

If you want to reuse a taught positions list from an existing project, please right-click **Point** and click **import points file**, as shown in Figure 1.39.

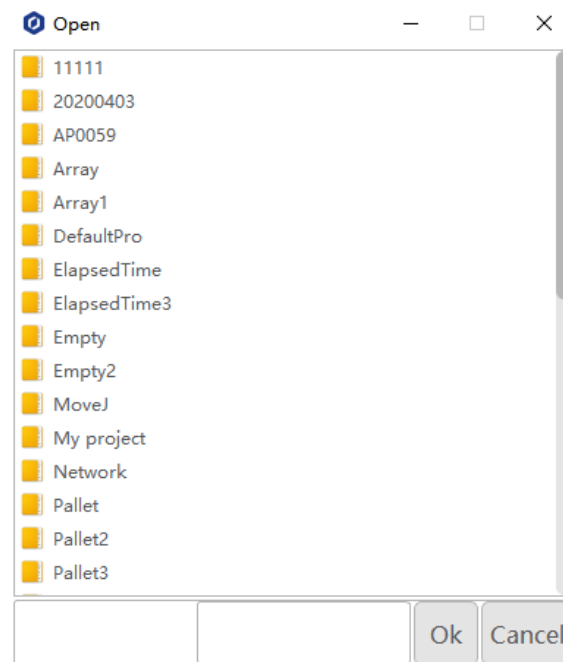


Figure 1.39 Import the existing teaching points list

1.5.3.2 Teaching points

Prerequisites

- The project has been created or imported
- DobotSCStudio has been in the manual mode


Procedure

After creating a project, please teach positions on the **TeachPoint** page for calling commands when programming a robot. If the existing taught positions list has been imported, this operation

can be skipped.

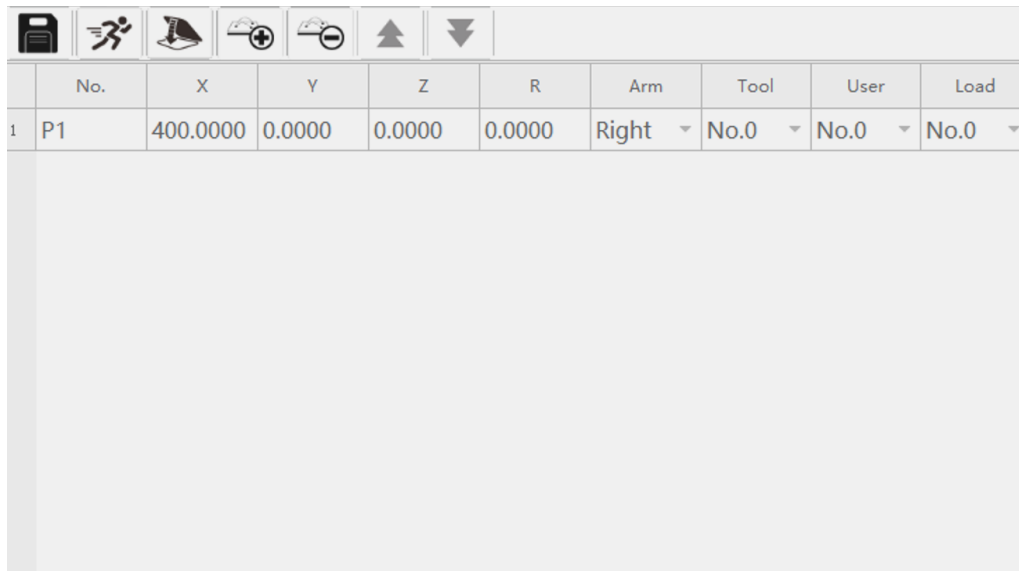
Step 1 Enable the robot motor.

Step 2 Click **Jog** buttons to move the robot to a point.

Step 3 Double click **Point** enter point page and click  to add a teaching point.

The teaching point information is displayed on the **TeachPoint** page, as shown in Figure 1.40.






Arm is the arm orientation. **Tool** is the Tool coordinate system and **User** is the User coordinate system.



No.	X	Y	Z	R	Arm	Tool	User	Load
1 P1	400.0000	0.0000	0.0000	0.0000	Right	No.0	No.0	No.0

Figure 1.40 Teaching points list of SCARA robot

Table 1.6 Button description

Button	Description
	Add a point
	Delete a point
	Cover a point. Select a teaching point, after jogging the robot to a point, click the icon to cover the selected teaching point
	Run to a point, select a point, click the button to run the robot to this point
	Save teaching point

Button	Description
	Previous page
	Next page

- You can select a taught position and double-click the parameters on the line to modify the relevant information, as shown in Figure 1.41.

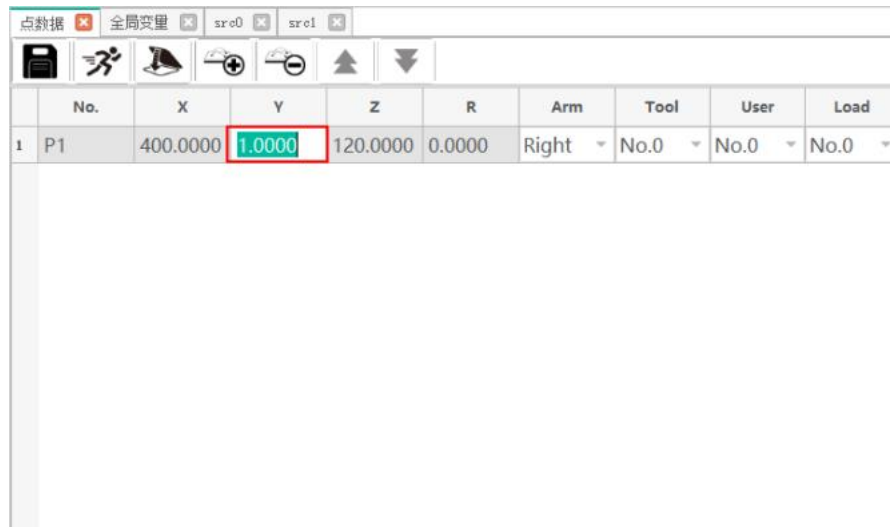


Figure 1.41 Modify the teaching point information

- Also, you can select a taught position and click to cover the current taught position.

Step 4 Add points by referring to Step 2 and Step 3.

Step 5 Click to save the teaching points.

NOTE

If the robot is a 6-axis type, the teach point page is as shown in Figure 1.42. **R, D, N, Cfg** indicate the arm parameters.

No.	X	Y	Z	Rx	Ry	Rz	R	D	N	Cfg	Tool	User	Load	
1	P1	400.00...	1.0000	120.00...	0.0000	0.0000	0.0000	-1 ▾	-1 ▾	-1 ▾	0	No.0 ▾	No.0 ▾	No.0 ▾
2	P2	-8.4353	105.02...	-136.0...	164.77...	0.0000	0.0000	-1 ▾	-1 ▾	-1 ▾	0	No.0 ▾	No.0 ▾	No.0 ▾

Figure 1.42 TeachPoint page

1.5.3.3 Writing a Program

Prerequisites

- The project has been created or imported.
- The points have been taught.

Procedure

In the robot system, we have encapsulated common commands for programming with Lua language. For details, please see *2 Program Language*.

Supposing that the **P1** and **P2** points have been taught on the **TeachPoint** page. We call **Go** command on the Src0 Page, to make the robot move between point P1 and point P2 circularly, as shown in Figure 1.43.

```

1 while ( true )
2 do
3 Go (P1)
4 Go (P2)
5 end
6
7
8
9

```

Figure 1.43 Lua program

Step 1 Click >Syntax, double click `while` to call the loop command, and set the loop

condition to **True**.

Step 2 Add the motion commands between **do** and **end**.

1. Click **Fx** > **Move**.

The motion commands list is displayed, as shown in Figure 1.44.

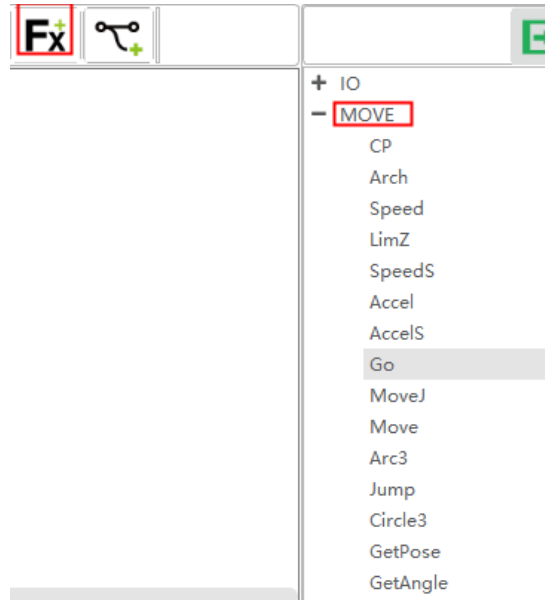


Figure 1.44 Motion commands list

2. Select a command from the motion commands list and click it on the edit window of the **Src0** page.

The parameter setting page of this command is displayed. Take the **Go** command as an example. You can set the point where the robot will move to in the **Go** mode.

3. Select **P1** on the **First Parameter** section of the Go command setting page, and then click **Insert**. Namely, the robot moves to **P1** point in the **Go** mode.

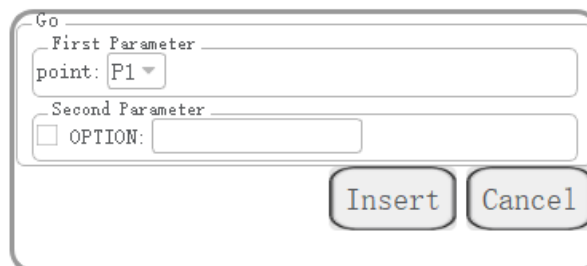


Figure 1.45 Call the **Go** command

If you want to set the motion speed, arm orientation, you can set them on the **Second Parameter** section, as shown in Figure 1.46.

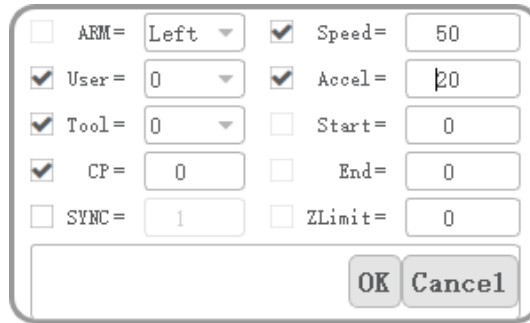


Figure 1.46 Set the optional parameters

4. Wrap and execute 2 again.
5. Select **P2** on the **First Parameter** section of the **Go** command setting page, and then click **Insert**. Namely, the robot moves to **P2** point in the **Go** mode.

NOTE

If you want to debug a robot program step by step, please set the breakpoint when writing the program. Click the right line to set, as shown in Figure 1.47.

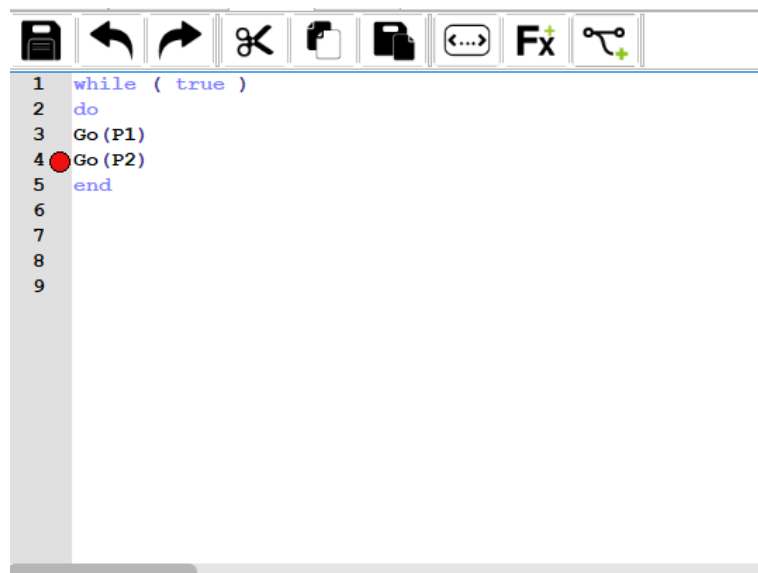


Figure 1.47 Set breakpoint

Step 3 Click .

Now, a simple program has been written.

1.5.3.4 Debugging Program

Step 1 Switch DobotSCStudio to the auto mode.

Step 2 Click to enable the motor.

Now, the programming page is as shown in Figure 1.48.

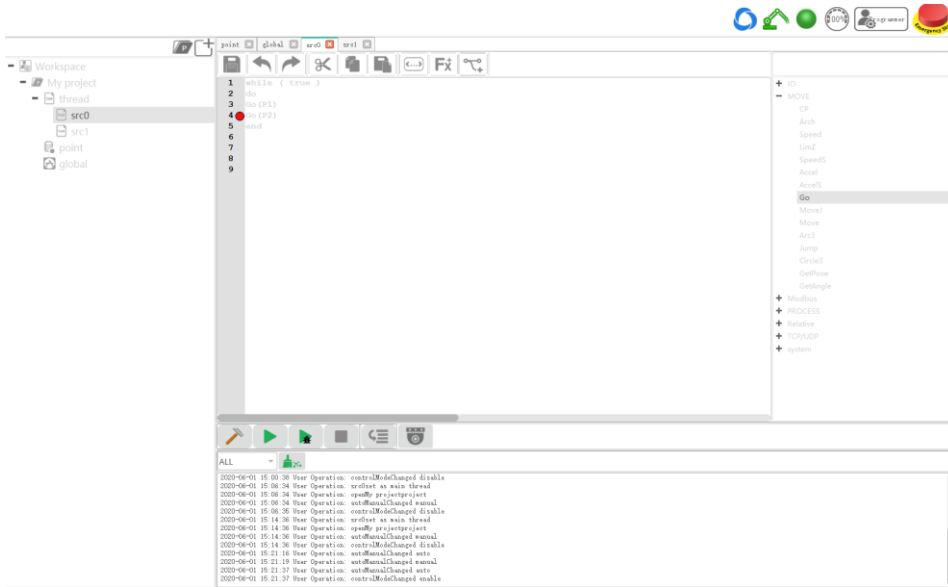



















Figure 1.48 Programming page







Table 1.7 lists the description of the program-running buttons which are shown in Figure 1.48

Table 1.7 Program-running button description







Icon	Description
	Build program Check if the code is correct
	Once-click run After clicking this button,  turns into  and the program starts running If you need to pause the running program, please click 
	Start to run a program Click once: Start to debug a program,  turns into  Click twice: Start to run a program,  turns into  If you need to pause the running program, please click 
	Stop the running program
	Step into

Icon	Description
	This button is valid only if  turns into 
	Monitor The debugging process can be monitored in real time while debugging the program


Step 3 Click  to start debugging the program.

- If you have set a breakpoint, the program will be run to the previous line of the breakpoint and then be stopped. If the program needs to be run again, please click  and then click .
- If you want to run a program step by step, please click . After  turns into , please click .

1.6 Enabling

- Enable the motor in the manual mode: Click  in manual mode. When the icon  turns into , you can jog the robot normally.
- Enable the motor in the auto mode: Click  in auto mode. When the icon  turns into , the robot arm can be controlled by running the program.

1.7 Setting Global Velocity Rate

Please click  and then click buttons to increase or decrease the global velocity ratio by 1%, 5% or 10% on the operation panel, or drag the slider directly to set the global velocity, as shown in Figure 1.49.

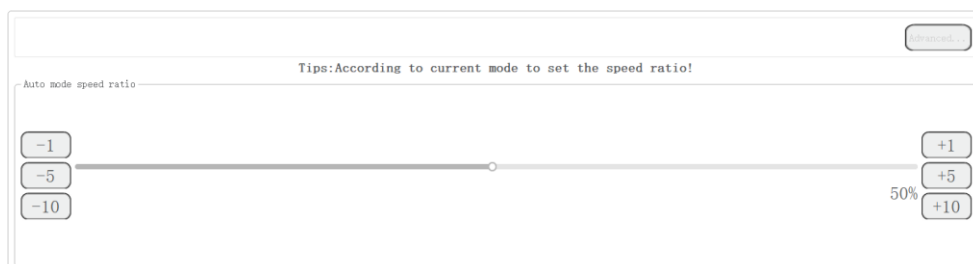


Figure 1.49 Modify the global velocity rate

When doing jogging or playback, the method calculating the velocity and acceleration for each axis (in Joint or Cartesian coordinate system) is shown as follows.



- Actual jogging velocity = the maximum jogging velocity * global velocity rate
- Actual jogging acceleration = the maximum jogging acceleration* global velocity rate
- Actual playback velocity = the maximum playback velocity * global velocity rate * the set velocity rate in the velocity function
- Actual playback acceleration = the maximum playback acceleration* global velocity rate * the set acceleration rate in the acceleration function
- Actual playback jerk = the maximum playback jerk * global velocity rate * the set acceleration rate in the jerk function

NOTE


- The maximum velocity, acceleration, or jerk can be set on the **Settings** page. For details, please see *1.2.1 Setting Motion Parameter*.
- The rates (velocity rate, acceleration rate, or jerk rate) can be set in the related speed functions. For details, please see *2.9 Motion Parameter Commands*.

1.8 Alarm Description

If teaching point is incorrect, for example, a robot moves to where a point is at a limited position or a singular point, an alarm will be triggered.

If an alarm is triggered when running a robot, the alarm icon  on the DobotSCStudio turns into . You can check the alarm information on the **Alarm** page, as shown in Figure 1.50.

Please clear the alarm as follows:

- If a limitation alarm is triggered, please jog the limited joint axis towards the opposite direction in the manual mode to clear the alarm.
- If other alarms are triggered, please click  in the manual mode on the alarm page to clear the alarm. If the alarm cannot be cleared, please reboot the robot.

	Id	Type	Level	Cause	Solution
1	0x45	Controller Error	0	Joint3 exceeds negative limit	

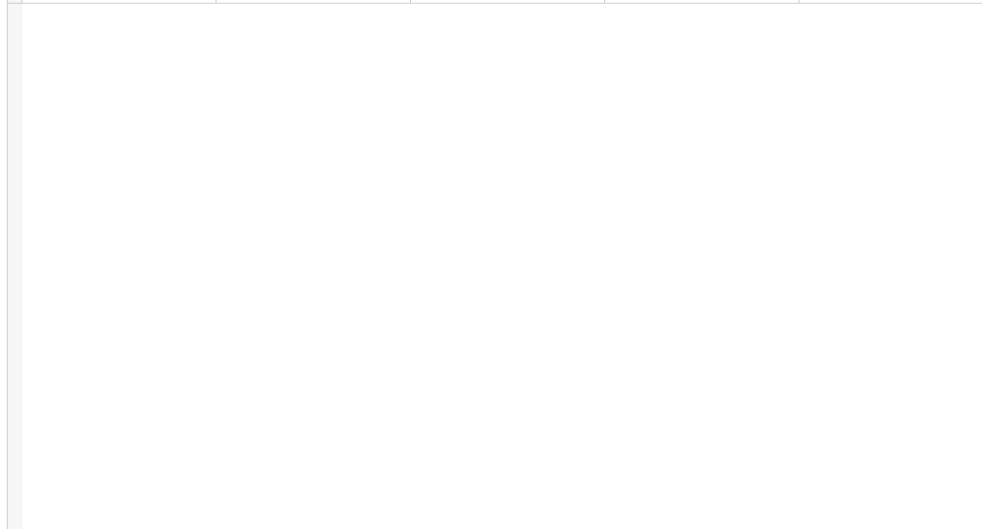


Figure 1.50 Alarm page

2. Program Language

SC series controller encapsulates the robot dedicated API commands for programming with Lua language. This section describes commonly used commands for reference.

2.1 Arithmetic Operators

Table 2.1 Arithmetic operator

Command	Description
+	Addition
-	Subtraction
*	Multiplication
/	Floating point division
//	Floor division
%	Remainder
^	Exponentiation
&	And operator
	OR operator
~	XOR operator
<<	Left shift operator
>>	Right shift operator

2.2 Relational Operator

Table 2.2 Relational Operator

Command	Description
==	Equal
~=	Not equal
<=	Equal or less than
>=	Equal or greater than
<	Less than
>	Greater than

2.3 Logical Operators

Table 2.3 Logical operator

Command	Description
or	Logical OR operator
not	Logical NOT operator
and	Logical AND operator

2.4 General Keywords

Table 2.4 General keyword

Command	Description
break	Break out of a loop
local	Define a local variable, which is available in the current script
nil	Null
return	Return a value
enter	Line feed

2.5 General Symbol

Table 2.5 General symbol

Command	Description
#	Get the length of the array table

2.6 Processing Control Commands

Table 2.6 Processing control command

Command	Description
if...then...else...elseif...end	Conditional instruction (if)
while...do...end	Loop instruction (while)
for...do...end	Loop instruction (for)
repeat... until()	Loop instruction (repeat)

2.7 Global Variable

The robot global variables can be defined in the **global.lua** file, including global functions, global points, and global variables.

- Global function:

```
function exam()
    print("This is an example")
end
```

- Global point:
 - SCARA robot: Define a Cartesian coordinate point, of which the arm orientation is right handy orientation, the User and Tool coordinate systems are both default coordinate systems.

```
P = {armOrientation = "right", coordinate = {400,0,0,0}, tool = 0, user = 0}
```

- Six-axis robot: Define a joint coordinate point, of which **R** sets to **1**, **D** sets to **-1**, **N** sets to **0**, **Cfg** sets to **1**, the User and Tool coordinate systems are both default coordinate systems.

```
P = {armOrientation = {1, 1, -1, 1}, joint = {20,10,22,2.14,0.87,3.85}, tool = 0, user = 0}
```

- Global variable

```
flag = 0
```

2.8 Motion Commands

Table 2.7 Motion command

Command	Description
Go	Move from the current position to a target position in a point-to-point mode under the Cartesian coordinate system
MoveJ	Move from the current position to a target position in a point-to-point motion under the Joint coordinate system
Move	Move from the current position to a target position in a straight line under the Cartesian coordinate system
Arc3	Move from the current position to a target position in an arc interpolated mode under the Cartesian coordinate system
Jump	<ul style="list-style-type: none"> • If the robot is a SCARA type, the robot moves from the current position to a target position in the Go mode. The trajectory looks like a door • If the robot is a six-robot type, the robot moves from the current position to a target position in the Move mode. The trajectory looks like a door

Command	Description
	•
Circle3	Move from the current position to a target position in a circular interpolated mode under the Cartesian coordinate system
RP	Set the X, Y, Z axes offset under the Cartesian coordinate system to return a new Cartesian coordinate point
RJ	Set the joint offset under the Joint coordinate system to return a new joint coordinate point
MoveR	Move from the current position to the offset position in a straight line under the Cartesian coordinate system
GoR	Move from the current position to the offset position in a point-to-point mode under the Cartesian coordinate system
MoveJR	Move from the current position to the offset position in a point-to-point motion under the Joint coordinate system

 NOTICE

Optional parameters for each motion command can be set individually

Table 2.8 Go command

Function	<code>Go(P,"ARM=Left User=1 Tool=2 CP=1 Speed=50 Accel=20 SYNC=1")</code>
Description	Move from the current position to a target position in a point-to-point mode under the Cartesian coordinate system

Parameter	<p>Required parameter: P: Indicate target point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0-9 • CP: Whether to set continuous path function. Value range: 0- 100 • Speed: Velocity rate. Value range: 1 - 100 • Accel: Acceleration rate. Value range: 1 -100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<ol style="list-style-type: none"> The robot moves to point P1 as the default setting <pre>Go(P1)</pre> The SCARA robot moves to point P1 as the righty hand orientation with 50% velocity rate and 50% acceleration rate <pre>Go(P1," ARM=Right Speed=50 Accel=50")</pre> Define a Cartesian coordinate point P1 and the SCARA robot moves to this point as the righty hand orientation with 50% velocity rate and 50% acceleration rate <pre>local P1 = {armOrientation = "right" ,coordinate={20,15,52,0} } Go(P1)</pre>

Table 2.9 MoveJ command

Function	<code>MoveJ(P," CP=1 Speed=50 Accel=20 SYNC=1")</code>
Description	Move from the current position to a target position in a point-to-point motion under the Joint coordinate system
Parameter	<p>Required parameter: P: Indicate the joint angle of the target point, which cannot be obtained from the TeachPoint page. You need to define the joint coordinate point before calling this command</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • CP: Whether to set continuous path function. Value range: 0 - 100 • Speed: Velocity rate. Value range: 1 - 100 • Accel: Acceleration rate. Value range: 1 - 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this

	command, it will not return until it is executed completely
Example	<pre>local P = {joint={0,-0.0674194,0,0}}</pre> <pre>MoveJ(P)</pre> <p>Define a joint coordinate point P and the SCARA robot moves to this point as the default setting</p>

Table 2.10 Move command

Function	<code>Move(P,"ARM=Left User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")</code>
Description	Move from the current position to a target position in a straight line under the Cartesian coordinate system
Parameter	<p>Required parameter: P: Indicate the target point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0 - 9 • CP: Whether to set continuous path function. Value range: 0 - 100 • SpeedS: Velocity rate. Value range: 1 - 100 • AccelS: Acceleration rate. Value range: 1 - 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<ol style="list-style-type: none"> 1. The robot moves to point P1 as the default setting <pre>Move(P1)</pre> 2. The SCARA robot moves to point P1 as the lefty hand orientation with 50% velocity rate and 50% acceleration rate <pre>Move(P1," ARM=Left SpeedS=50 AccelS=20")</pre> 3. Define a Cartesian coordinate point P1 and the SCARA robot moves to this point as the default setting <pre>local P1 = {coordinate={20,15,52,0}}</pre> <pre>Move(P1)</pre>

Table 2.11 Arc3 command

Function	<code>Arc3(P1,P2, "ARM=Left User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")</code>
----------	--

Description	<p>Move from the current position to a target position in an arc interpolated mode under the Cartesian coordinate system</p> <p>This command needs to combine with other motion commands, to obtain the starting point of an arc trajectory</p>
Parameter	<p>Required parameter:</p> <ul style="list-style-type: none"> • P1: Middle point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported • P2: End point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported <p>Optional parameter:</p> <ul style="list-style-type: none"> • Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0 - 9 • CP: Whether to set continuous path function. Value range: 0 - 100 • SpeedS: Velocity rate. Value range: 1 - 100 • AccelS: Acceleration rate. Value range: 1 - 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<pre>While true do Go(P1) Arc3(P2,P3) end</pre> <p>The robot cycles from point P1 to point P3 in the arc interpolated mode</p>

Table 2.12 Jump command

Function	<pre>Jump(P,"ARM=Left User=1 Tool=2 Speed=50 Accel=20 Arch=1 SYNC=1") Jump(P,"ARM=Left User=1 Tool=2 Speed=50 Accel=20 Start=10 ZLimit=80 End=50 SYNC=1")</pre>
Description	<ul style="list-style-type: none"> • If the robot is a SCARA type, the robot moves from the current position to a target position in the Go mode. The trajectory looks like a door • If the robot is a six-robot type, the robot moves from the current position to a target position in the Move mode. The trajectory looks like a door

Parameter	<p>Required parameter: P: Indicate the target point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported. Also, the target point cannot be higher than ZLimit, to avoid an alarm about JUMP parameter error</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0 - 9 • Speed: Velocity rate. Value range: 1 - 100 • Accel: Acceleration rate. Value range: 1 - 100 • Arch: Arch index. Value range: 0 - 9 • Start: Lifting height • ZLimit: Maximum lifting height • End: Dropping height • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<p>Jump(P1)</p> <p>The robot moves to point P1 in the Jump mode</p>

 NOTICE

The lifting height and dropping height cannot be higher than ZLimit, to avoid an alarm on JUMP parameter error.

Table 2.13 Circle3 command

Function	Circle3(P1,P2,Count, "ARM=Left User=1 Tool=2 CP=1SpeedS=50 AccelS=20")
Description	<p>Move from the current position to a target position in a circular interpolated mode under the Cartesian coordinate system</p> <p>This command needs to combine with other motion commands, to obtain the starting point of an arc trajectory</p>
Parameter	<p>Required parameter</p> <ul style="list-style-type: none"> • P1: Middle point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported • P2: End point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported

	<ul style="list-style-type: none"> Count: Number of circles. Value range: 1 - 999 <p>Optional parameter:</p> <ul style="list-style-type: none"> ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid User: Indicate User coordinate system. Value range: 0 - 9 Tool: Indicate Tool coordinate system. Value range: 0 - 9 CP: Whether to set continuous path function. Value range: 0 - 100 SpeedS: Velocity rate. Value range: 1 - 100 AccelS: Acceleration rate. Value range: 1 - 100 SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<p>Go(P1)</p> <p>Circle3(P2,P3,1)</p> <p>Robot cycles from point P1 to point P3 in the circular interpolated mode</p>

Table 2.14 RP command

Function	RP(P1, {OffsetX, OffsetY, OffsetZ})
Description	<p>Set the X, Y, Z axes offset under the Cartesian coordinate system to return a new Cartesian coordinate point</p> <p>The robot can move to this point in all motion commands except MoveJ</p>
Parameter	<ul style="list-style-type: none"> P1: Indicate the current Cartesian coordinate point, which is user-defined or obtained from the TeachPoint page. Only Cartesian coordinate points are supported OffsetX, OffsetY, OffsetZ: X, Y, Z axes offset in the Cartesian coordinate system Unit: mm
Return	Cartesian coordinate point
Example	<p>P2=RP(P1, {50,10,32})</p> <p>Move(P2) or Move(RP(P1, {50,10,32}))</p>

Table 2.15 RJ command

Function	RJ(P1, {Offset1, Offset2, Offset3, Offset4, Offset5, Offset6})
Description	<p>Set the joint offset in the Joint coordinate system to return a new joint coordinate point</p> <p>The robot can move to this point only in MoveJ command</p>

Parameter	<ul style="list-style-type: none"> P1: Indicate the current joint coordinate point, which cannot be obtained from the TeachPoint page. You need to define the joint coordinate point before calling this command Offset1~Offset6: J1 - J6 axes offset. If the robot is a SCARA type, Offset5 and Offset 6 are invalid <p>Unit: °</p>
Return	Joint coordinate point
Example	<p>Take a SCARA robot as an example:</p> <pre>local P1 = {joint={0,-0.0674194,0,0}}</pre> <pre>P2=RJ(P1, {60,50,32,30})</pre> <pre>MoveJ(P2) or MoveJ(RJ(P1, {60,50,32,30}))</pre>

Table 2.16 GoR command

Function	<code>GoR({OffsetX, OffsetY, OffsetZ}, "ARM=Left User=1 Tool=2 CP=1 Speed=50 Accel=20 SYNC=1 ")</code>
Description	Move from the current position to the offset position in a point-to-point mode under the Cartesian coordinate system
Parameter	<p>Required parameter: OffsetX, OffsetY, OffsetZ: X, Y, Z axes offset in the Cartesian coordinate system</p> <p>Unit: mm</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid User: Indicate User coordinate system. Value range: 0 - 9 Tool: Indicate Tool coordinate system. Value range: 0-9 CP: Whether to set continuous path function. Value range: 0- 100 Speed: Velocity rate. Value range: 1 - 100 Accel: Acceleration rate. Value range: 1 -100 SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<pre>Go(P1)</pre> <pre>GoR({10,10,10}, "Accel=100 Speed=100 CP=100")</pre>

Table 2.17 MoveJR command

Function	<code>MoveJR({Offset1, Offset2, Offset3, Offset4, Offset5, Offset6}," CP=1 Speed=50 Accel=20 SYNC=1")</code>
Description	Move from the current position to the offset position in a point-to-point motion under the Joint coordinate system
Parameter	<p>Required parameter: Offset1 - Offset6: J1 - J6 axes offset. If the robot is a SCARA type, Offset5 and Offset 6 are invalid</p> <p>Unit: °</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • CP: Whether to set continuous path function. Value range: 0 - 100 • Speed: Velocity rate. Value range: 1 - 100 • Accel: Acceleration rate. Value range: 1 - 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	<pre>Go(P1) MoveJR({20,20,10,0},"SYNC=1")</pre>

Table 2.18 MoveR command

Function	<code>MoveR({OffsetX, OffsetY, OffsetZ},"ARM=Left User=1 Tool=2 CP=1 SpeedS=50 AccelS=20 SYNC=1")</code>
Description	Move from the current position to the offset position in a straight line under the Cartesian coordinate system
Parameter	<p>Required parameter: OffsetX, OffsetY, OffsetZ: X, Y, Z axes offset in the Cartesian coordinate system</p> <p>Unit: mm</p> <p>Optional parameter:</p> <ul style="list-style-type: none"> • ARM: Arm orientation: If the robot is a SCARA type, please set to left or right. If the robot is a 6-axis type, this parameter is invalid • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0 - 9 • CP: Whether to set continuous path function. Value range: 0 - 100 • SpeedS: Velocity rate. Value range: 1 - 100 • AccelS: Acceleration rate. Value range: 1 - 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous

	execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1 , it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Example	Go(P1) MoveR({20,20,20},"AccelS=100 SpeedS=100 CP=100")

2.9 Motion Parameter Commands

Table 2.19 Motion parameter command

Command	Description
Accel	Set the acceleration rate. This command is valid only when the motion mode is Go , Jump , or MoveJ
AccelS	Set the acceleration rate. This command is valid only when the motion mode is Move , Arc3 , or Circle3
Speed	Set the velocity rate. This command is valid only when the motion mode is Go , Jump , or MoveJ
SpeedS	Set the velocity rate. This command is valid only when the motion mode is Move , Arc3 , or Circle3
Arch	Set the index of sets of parameters (StartHeight , zLimit , EndHeight) in Jump mode
CP	Set the continuous path function
LimZ	Set the maximum lifting height in the Jump mode

Table 2.20 Accel command

Function	Accel(R)
Description	Set the acceleration rate. This command is valid only when the motion mode is Go , Jump , or MoveJ
Parameter	R: Percentage. Value range: 1 - 100
Example	Accel(50) Go(P1) The robot moves to point P1 with 50% acceleration rate

Table 2.21 AccelS command

Function	AccelS(R)
Description	Set the acceleration rate. This command is valid only when the motion mode is Move , Arc3 , or Circle3
Parameter	R: Percentage. Value range: 1 - 100
Example	<pre>AccelS(20) Move(P1)</pre> <p>The robot moves to point P1 with 20% acceleration rate</p>

Table 2.22 Speed command

Function	Speed(R)
Description	Set the velocity rate. This command is valid only when the motion mode is Go , Jump , or MoveJ
Parameter	R: Percentage. Value range: 1 - 100
Example	<pre>Speed(20) Go(P1)</pre> <p>The robot moves to point P1 with 20% velocity rate</p>

Table 2.23 SpeedS command

Function	SpeedS(R)
Description	Set the acceleration rate. This command is valid only when the motion mode is Move , Arc3 , or Circle3
Parameter	R: Percentage. Value range: 1 - 100
Example	<pre>SpeedS(20) Move(P1)</pre> <p>The robot moves to point P1 with 20% velocity rate</p>

Table 2.24 Arch command

Function	Arch(Index)
Description	<p>Set the index of sets of parameters (StartHeight, zLimit, EndHeight) in the Jump mode</p> <p>The sets of parameters need to be set on the Config> RobotParams>PlayBack Arch of DobotSCStudio. For details, please see <i>1.2.1 Setting Motion Parameter</i></p>
Parameter	Index: Index of the sets parameters. Value range: 0 - 9

	This parameter is valid only when the right index has been selected from the Config>RobotParams>PlayBack Arch of DobotSCStudio
Example	Arch(1) Jump(P1)

Table 2.25 CP command

Function	CP(R)
Description	Set the continuous path rate. This command is valid only when the motion mode is Go , Move , Arc3 , Circle3 , or MoveJ
Parameter	R: Continuous path rate. Value range: 0 -100 0 indicates that the Continuous path function is disabled
Example	CP(50) Move(P1) Move(P2) The robot moves from point P1 to point P2 with 50% Continuous path ratio

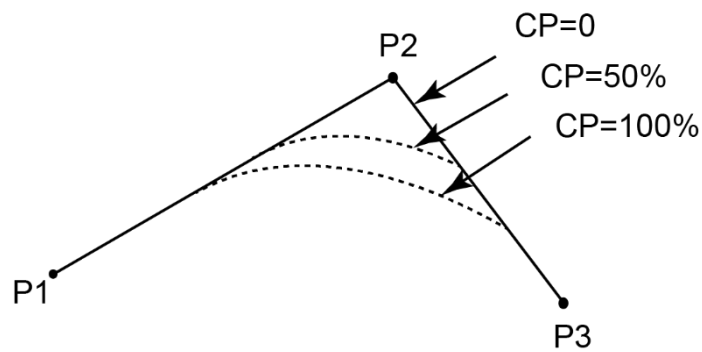


Figure 2.1 Continuous path

Table 2.26 LimZ command

Function	LimZ(zValue)
Description	Set the maximum lifting height in Jump mode
Parameter	zValue: The maximum lifting height which cannot exceed the Z-axis limiting position of the robot
Example	LimZ(80) Jump(P," Start=10 ZLimit=LimZ End=50")

2.10 Input/output Commands

Table 2.27 Input/output command

Command	Description
DI	Get the status of the digital input port
DO	Set the status of the digital output port (Queue command)
DOExecute	Set the status of the digital output port (Immediate command)

NOTE

Dobot robot system supports two kinds of commands: Immediate command and queue command:

- Immediate command: The robot system will process the command once received regardless of whether there is the rest commands processing or not in the current controller;
- Queue command: When the robot system receives a command, this command will be pressed into the internal command queue. The robot system will execute commands in the order in which the commands were pressed into the queue.

Table 2.28 Digital input command

Function	<code>DI(index)</code>
Description	Get the status of the digital input port
Parameter	index: Digital input index. Value range: 1 - 16
Return	<ul style="list-style-type: none"> • When an index is set in the DI function, DI(index) returns the status (ON/OFF) of this specified input port • When there is no index in the DI function, DI() returns the status of all the input ports, which are saved in a table <p>For example, local di=(), the saving format is {num = 24 value = {0x55, 0xAA, 0x52}}, you can obtain the status of the specified input port with di.num and di.value[n]</p>
Example	<pre>if (DI(1))==ON then Move(P1) end</pre> <p>The robot moves to point P1 when the status of the digital input port 1 is ON</p>

Table 2.29 Digital output command (Queue command)

Function	<code>DO(index, ON OFF)</code>
Description	Set the status of digital output port (Queue command)
Parameter	<ul style="list-style-type: none"> index: Digital output index. Value range: 1 - 24 ON/OFF: Status of the digital output port. ON: High level; OFF: Low level
Example	<code>DO(1,ON)</code> Set the status of the digital output port 1 to ON

Table 2.30 Digital output command (Immediate command)

Function	<code>DOExecute(index, ON OFF)</code>
Description	Set the status of digital output port (Immediate command)
Parameter	<ul style="list-style-type: none"> index: Digital output index. Value range: 1 - 24 ON/OFF: Status of the digital output port. ON: High level; OFF: Low level
Example	<code>DOExecute(1,OFF)</code> Set the status of the digital output port 1 to OFF

2.11 Program Managing Commands

Table 2.31 Program managing command

Command	Description
Wait	Set the delay time for robot motion commands
Sleep	Set the delay time for all commands
Pause	Pause the running program
ResetElapsedTime	Start timing
ElapsedTime	Stop timing
System	Get the current time

Table 2.32 Wait command




Function	<code>Wait(time)</code>
Description	Set the delay time for robot motion commands
Parameter	time: Delay time. Unit: ms

Example	<pre>Go(P1) Wait(1000) Wait for 1000ms after the robot moves to point P1</pre>
---------	---

Table 2.33 Sleep command

Function	<code>Sleep(<i>time</i>)</code>
Description	Set the delay time for all commands
Parameter	time: Delay time. Unit: ms
Example	<pre>while true do Speed(100) Go(P1) sleep(3) Speed(100) Accel(40) Go(P2) sleep(3) end</pre>

Table 2.34 Pause command

Function	<code>Pause()</code>
Description	<p>Pause the running program</p> <p>When the program runs to this command, robot pauses running and the button  on DobotSCStudio turns into . If the robot continues to run, please click .</p>
Parameter	None

Example	<pre> while true do Go(P1) Go(P2) Pause() Go(P3) Go(P4) end </pre> <p>The robot moves to point P2 and then pauses running</p>
---------	---

Table 2.35 Star timing command

Function	ResetElapsedTime()
Description	Start timing after all commands before this command are executed completely. Use in conjunction with ElapsedTime() command For example: Get the execution time that a piece of code takes
Parameter	None
Return	None
Example	<pre> Go(P2, " Speed=100 Accel=100") ResetElapsedTime() for i=1,10 do Jump(P1, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") Jump(P2, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") end print (ElapsedTime()) Sleep(1000) </pre>

Table 2.36 Stop timing command

Function	ElapsedTime()
Description	Stop timing and return the time difference. Use in conjunction with ResetElapsedTime() command
Parameter	None
Return	Time difference. Unit: ms

Example	<pre> Go(P2, " Speed=100 Accel=100") ResetElapsedTime() for i=1,10 do Jump(P1, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") Jump(P2, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") end print (ElapsedTime()) Sleep(1000) </pre>
---------	--

Table 2.37 Get current time command

Function	Systime()
Description	Get the current time
Parameter	None
Return	Current time
Example	<pre> Go(P2, " Speed=100 Accel=100") local time1=Systime() for i=1,10 do Jump(P1, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") Jump(P2, " Speed=100 Accel=100 Start=0 End=0 ZLimit=185") end local time2=Systime() local time = time2 - time1 Sleep(1000) </pre>

2.12 Pose Getting Command

Table 2.38 Pose command (1)

Function	GetPose()
Description	Get the current pose of the robot under the Cartesian coordinate system If you have set the User or Tool coordinate system, the current pose is under the current User or Tool coordinate system
Parameter	None
Return	Cartesian coordinate of the current pose

Example	<pre> local currentPose = GetPose() --Get the current pose local liftPose = {armOrientation = left , coordinate = {currentPose.coordinate[1], currentPose. coordinate[2], currentPose. coordinate[3],currentPose. coordinate[4] }, tool = currentPose.tool, user = currentPose.user } -- Lift a certain height Go(liftPose,"Speed=100 Accel=100") Go(P1) </pre>
---------	---

Table 2.39 Pose command (2)

Function	GetAngle()
Description	Get the current pose of the robot under the Joint coordinate system
Parameter	None
Return	Joint coordinate of the current pose
Example	<pre> local armPose local joint = GetAngle() --Get the current pose if joint.joint[2] > 0 then armPose = "right" else armPose = "left" end local liftPose = {armOrientation = armPose , joint = {joint.joint[1], joint.joint[2], joint.joint[3], joint.joint[4]}, tool = 0, user = 0} </pre>

2.13 TCP

Table 2.40 Create TCP command

Function	err, socket = TCPCreate(isServer, IP, port)
Description	Create a TCP network Only support a single connection

Parameter	<p>isServer: Whether to create a server. 0: Create a client; 1: Create a server</p> <p>IP: IP address of the server, which is in the same network segment of the client without conflict</p> <p>port: Server port</p> <p>When the robot is set as a server, port cannot be set to 502 and 8080. Otherwise, it will be in conflict with the Modbus default port or the port used in the conveyor tracking application, causing the creation to fail</p>
Return	<p>err:</p> <p>0: TCP network is created successfully</p> <p>1: TCP network is created failed</p> <p>Socket: Socket object</p>
Example	Please refer to Program 2.1 and Program 2.2

Table 2.41 TCP connection command

Function	<code>TCPStart(socket, timeout)</code>
Description	Connect a client to a server with the TCP protocol
Parameter	<p>socket: Socket object</p> <p>timeout: Wait timeout. Unit: s. If timeout is 0, the connection is still waiting. If not, after exceeding the timeout, the connection is exited.</p>
Return	<ul style="list-style-type: none"> • 0: TCP connection is successful • 1: Input parameters are incorrect • 2: Socket object is not found • 3: Timeout setting is incorrect • 4: If the robot is set as a client, it indicates that the connection is wrong. If the robot is set as a server, it indicates that receiving data is wrong
Example	Please refer to Program 2.1 and Program 2.2

Table 2.42 Receive data command

Function	<code>err, Recbuf = TCPRead(socket, timeout, type)</code>
Description	<p>Robot as a client receives data from a server</p> <p>Robot as a server receives data from a client</p>

Parameter	socket: socket object timeout: Receiving timeout. Unit: s. If timeout is 0 or is not set, this command is a block reading. Namely, the program will not continue to run until receiving data is complete. If not, after exceeding the timeout, the program will continue to run regardless of whether receiving data is complete type: Buffer type. If type is not set, the buffer format of RecBuf is a table. If type is set to string , the buffer format is a string
Return	err: 0: Receiving data is successful 1: Receiving data is failed Recbuf: Data buffer
Example	Please refer to Program 2.1 and Program 2.2

Table 2.43 Send data command

Function	TCPWrite(socket, buf, timeout)
Description	The robot as a client sends data to a server The robot as a server sends data to a client
Parameter	socket: Socket object buf: Data sent by the robot timeout: Timeout. Unit: s. If timeout is 0 or not set, this command is a block reading. Namely, the program will not continue to run until sending data is complete. If not, after exceeding the timeout, the program will continue to run regardless of whether sending data is complete
Return	0: Sending data is successful 1: Sending data is failed
Example	Please refer to Program 2.1 and Program 2.2

Table 2.44 Release TCP network command

Function	TCPDestroy(socket)
Description	Release a TCP network
Parameter	socket: Socket object
Return	0: Releasing TCP is successful 1: Releasing TCP is failed
Example	Please refer to Program 2.1 and Program 2.2


NOTICE

- Only a single TCP connection is supported. Please start the server before connecting a client. Please shut down the client before disconnection, to avoid re-connection failure since the server port is not released in time.
- When the robot is set as a server, the IP address of the robot can be checked and modified on the **Config> NetworkSetting** page of DobotSCStudio. Also, the port cannot be set to **502** and **8080**. Otherwise, it will be in conflict with the Modbus default port or the port used in the conveyor tracking application, causing the creation to fail.

Program 2.1 TCP server demo

```

local ip="192.168.5.1"                // IP address of the robot as a server
local port=6001                       // Server port
local err=0
local socket=0
err, socket = TCPCreate(true, ip, port)
if err == 0 then
    err = TCPStart(socket, 0)
    if err == 0 then
        local RecBuf
        while true do
            TCPWrite(socket, "tcp server test")        // Server sends data to client
            err, RecBuf = TCPRead(socket,0,"string")    // Server receives the data from client
            if err == 0 then
                Go(P1)                                //Start to run motion commands after the server receives data
                Go(P2)
                print(Recbuf)
            else
                print("Read error ".. err)
                break
            end
        end
    end
else
    print("Create failed ".. err)
end
TCPDestroy(socket)
else

```

```
    print("Create failed ".. err)
end
```

Program 2.2 TCP client demo

```
local ip="192.168.5.25"           // External equipment such as a camera is set as the server
local port=6001                 // Server port
local err=0
local socket=0
err, socket = TCPCreate(false, ip, port)
if err == 0 then
    err = TCPStart(socket, 0)
    if err == 0 then
        local RecBuf
        while true do
            TCPWrite(socket, "tcp client test")           // Client sends data to server
            TCPWrite(socket, {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07})
            err, RecBuf = TCPRead(socket, 0)              // Client receives data from server
            if err == 0 then
                Go(P1)           // Start to run motion commands after the client receives the data
                Go(P2)
                print(Recbuf)
            else
                print("Read error ".. err)
                break
            end
        end
    end
    else
        print("Create failed ".. err)
    end
    TCPDestroy(socket)
else
    print("Create failed ".. err)
end
```

2.14 UDP

Table 2.45 Create UDP network command

Function	<code>err, socket = UDPCreate(isServer, IP, port)</code>
Description	Create a UDP network Only a single connection is supported
Parameter	isServer: Whether to create a server. 0: Create a client; 1: Create a server IP: IP address of the server, which is in the same network segment of the client without conflict port: Server port When the robot is set as a server, port cannot be set to 502 or 8080. Otherwise, it will be in conflict with the Modbus default port or the port used in the conveyor tracking application, causing the creation to fail
Return	err: 0: The UDP network is created successfully 1: The UDP network is created failed socket: Socket object
Example	Please refer to Program 2.3 and Program 2.4

Table 2.46 Receive data command

Function	<code>err, Recbuf = UDPRead(socket, timeout, type)</code>
Description	The robot as a client receives data from a server The robot as a server receives data from a client
Parameter	socket: socket object timeout: Receiving timeout. Unit: s. If timeout is 0 or not set, this command is a block reading. Namely, the program will not continue to run until receiving data is complete. If not, after exceeding the timeout, the program will continue to run regardless of whether receiving data is complete type: Buffer type. If type is not set, the buffer format of RecBuf is a table. If type is set to string , the buffer format is a string
Return	err: 0: Receiving data is successful 1: Receiving data is failed Recbuf: Data buffer
Example	Please refer to Program 2.3 and Program 2.4

Table 2.47 Send data command

Function	<code>UDPWrite(socket, buf, timeout)</code>
Description	The robot as a client sends data to a server The robot as a server sends data to a client
Parameter	socket: Socket object buf: Data sent by the robot timeout: Timeout. Unit: s. If timeout is 0 or not set, this command is a block reading. Namely, the program will not continue to run until sending data is complete. If not, after exceeding the timeout, the program will continue to run regardless of whether sending data is complete
Return	0: Sending data is successful 1: Sending data is failed
Example	Please refer to Program 2.3 and Program 2.4

 NOTICE

- Only a single UDP connection is supported. Please start the server before connecting a client. Please shut down the client before disconnection, to avoid re-connection failure since the server port is not released in time.
- When the robot is set as a server, the IP address of the robot can be checked and modified on the **Config > NetworkSetting** page of DobotSCStudio. Also, the port cannot be set to **502** and **8080**. Otherwise, it will be in conflict with the Modbus default port or the port used in the conveyor tracking application, causing the creation to fail.

Program 2.3 UDP server demo

```

local ip="192.168.5.1" // IP address of the robot as a server
local port=6201 // Server port
local err=0
local socket=0
err, socket = UDPCreate(true, ip, port)
if err == 0 then
    local RecBuf
    while true do
        UDPWrite(socket, "udp server test") // Server sends data to client
        err, RecBuf = UDPRead(socket, 0) //Server receives the data from client
        if err == 0 then
            Go(P1) // Start to run motion commands after the server receives the data
        end
    end
end
    
```

```

        Go(P2)
        print(Recbuf)
    else
        print("Read error ".. err)
        break;
    end
end
else
    print("Create failed ".. err)
end

```

Program 2.4 UDP client demo

```

local ip="192.168.1.25" // IP address of the external equipment
as a server
local port=6200 // server port
local err=0
local socket=0

err, socket = UDPCreate(false, ip, port)
if err == 0 then
    local RecBuf
    while true do
        UDPWrite(socket, "udp client test") // Client sends data to server
        UDPWrite(socket, {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07})
        err, RecBuf = UDPRead(socket, 0) // Client receives the data from server
        if err == 0 then
            Go(P1) // Start to run motion commands after the client receives the data
            Go(P2)
            print(Recbuf)
        else
            print("Read error ".. err)
            break
        end
    end
end
else
    print("Create failed ".. err)
end

```

end

2.15 Modbus

2.15.1 Modbus Register Description

Modbus protocol is a serial communication protocol. The robot system can communicate with external equipment by this protocol. Here, External equipment such as a PLC is set as the Modbus master, and the robot system is set as the slave.

Modbus data is most often read and written as registers. Based on our robot memory space, we also define four types of registers: coil, discrete input, input, and holding registers for data interaction between the external equipment and robot system. Each register has 4096 addresses. For details, please see as follows.

- Coil register

Table 2.48 Coil register description

Coil register address (e.g.: PLC)	Coil register address (Robot system)	Data type	Description
00001	0	Bit	Start
00002	1	Bit	Pause
00003	2	Bit	Continue
00004	3	Bit	Stop
00005	4	Bit	Emergency stop
00006	5	Bit	Clear alarm
00007-0999	6-998	Bit	Reserved
01001-04096	999-4095	Bit	User-defined

- Discrete input register

Table 2.49 Discrete input register description

Discrete input register address (e.g.: PLC)	Discrete input register address(Robot system)	Data type	Description
10001	0	Bit	Automated exit
10002	1	Bit	Ready state
10003	2	Bit	Paused state
10004	3	Bit	Running state
10005	4	Bit	Alarm state
10006-10999	5-998	Bit	Reserved

Discrete input register address (e.g.: PLC)	Discrete input register address(Robot system)	Data type	Description
11000-14096	999-4095	Bit	User-defined

- Input register

Table 2.50 Input register description

Input register address (e.g.: PLC)	Input register address (Robot system)	Data type	Description
30001-34096	0-4095	Byte	Reserved

- Holding register

Table 2.51 Holding register description

Holding register address (e.g.: PLC)	Holding register address (Robot system)	Data type	Description
40001-41000	0-999	Byte	Reserved
41001-44095	1000-4095	Byte	User-defined

2.15.2 Command Description

Table 2.52 Rea coil register command

Function	<code>GetCoils(addr, count)</code>
Description	Read the coil value from the Modbus slave
Parameter	addr: Starting address of the coils to read. Value range: 0 - 4095 count: Number of the coils to read. Value range: 0 to 4096-addr
Return	Return a table, each with the value 1 or 0, where the first value in the table corresponds to the coil value at the starting address
Example	Read 5 coils starting at address 0 <code>Coils = GetCoils(0,5)</code> Return: <code>Coils={ 1,0,0,0,0}</code> As shown in Table 2.47, it indicates that the robot is in the starting state

Table 2.53 Set coil register command

Function	<code>SetCoils(addr, count, table)</code>
Description	Set the coil register in the Modbus slave This command is not supported when the coil register address is from 0 to 5
Parameter	Addr: Starting address of the coils to set. Value range: 6 - 4095 count: Number of the coils to set. Value range: 0 to 4096-addr table: Coil value, stored in a table
Return	None
Example	Set 5 coils starting at address 1024 local Coils = {0,1,1,1,0} SetCoils(1024, #coils, coils)

Table 2.54 Read discrete input register command

Function	<code>GetInBits(addr, count)</code>
Description	Read the discrete input value from Modbus slave
Parameter	addr: Starting address of the discrete inputs to read. Value range: 0-4095 count: Number of the discrete inputs to read. Value range: 0 to 4096-addr
Return	Return a table, each with the value 1 or 0, where the first value in the table corresponds to the discrete value at the starting address
Example	Read 5 discrete inputs starting at address 0 inBits = GetInBits(0,5) Return: inBits = {0,0,0,1,0} As shown in Table 2.48, it indicates the robot is in running state

Table 2.55 Read input register command

Function	<code>GetInRegs(addr, count, type)</code>
Description	Read the input register value with the specified data type from the Modbus slave

Parameter	<p>addr: Starting address of the input registers. Value range: 0 - 4095</p> <p>count: Number of the input registers to read. Value range: 0 ~ 4096-addr</p> <p>type: Data type</p> <ul style="list-style-type: none"> • Empty: Read 16-bit unsigned integer (two bytes, occupy one register) • “U16”: Read 16-bit unsigned integer (two bytes, occupy one register) • “U32”: Read 32-bit unsigned integer (four bytes, occupy two registers) • “F32”: Read 32-bit single-precision floating-point number (four bytes, occupy two registers) • “F64”: Read 64-bit double-precision floating-point number (eight bytes, occupy four registers)
Return	Return a table, the first value in the table corresponds to the input register value at the starting address
Example	<p>Example 1: Read a 16-bit unsigned integer starting at address 2048</p> <pre style="background-color: #f0f0f0;">data = GetInRegs(2048,1)</pre> <p>Example 2: Read a 32-bit unsigned integer starting at address 2048</p> <pre style="background-color: #f0f0f0;">data = GetInRegs(2048, 1, “U32”)</pre>

Table 2.56 Read holding register command

Function	GetHoldRegs(addr, count, type)
Description	Read the holding register value from the Modbus slave according to the specified data type
Parameter	<p>addr: Starting address of the holding registers. Value range: 0 - 4095</p> <p>count: Number of the holding registers to read. Value range: 0 to 4096-addr</p> <p>type: Datatype</p> <ul style="list-style-type: none"> • Empty: Read 16-bit unsigned integer (two bytes, occupy one register) • “U16”: Read 16-bit unsigned integer (two bytes, occupy one register) • “U32”: Read 32-bit unsigned integer (four bytes, occupy two registers) • “F32”: Read 32-bit single-precision floating-point number (four bytes, occupy two registers) • “F64”: Read 64-bit double-precision floating-point number (eight bytes, occupy four registers)
Return	Return a table, the first value in the table corresponds to the input register value at the starting address

Example	Example 1: Read a 16-bit unsigned integer starting at address 2048 <pre>data = GetHoldRegs(2048,1)</pre>
	Example 1: Read a 32-bit unsigned integer starting at address 2048 <pre>data = GetInRegs(2048, 1, "U32")</pre>

Table 2.57 Set holding register command

Function	<code>SetHoldRegs(addr, count, table, type)</code>
Description	Set the holding register in the Modbus slave
Parameter	<p>addr: Starting address of the holding registers to set. Value range: 0 - 4095</p> <p>count: Number of the holding registers to set. Value range: 0 to 4096-addr</p> <p>table: Holding register value, stored in a table</p> <p>type: Datatype</p> <ul style="list-style-type: none"> • Empty: Read 16-bit unsigned integer (two bytes, occupy one register) • "U16": Set 16-bit unsigned integer (two bytes, occupy one register) • "U32": Set 32-bit unsigned integer (four bytes, occupy two registers) • "F32": Set 32-bit single-precision floating-point number (four bytes, occupy two registers) • "F64": Set 64-bit double-precision floating-point number (eight bytes, occupy four registers)
Return	None
Example	<p>Example1: Set a 16-bit unsigned integer starting at address 2048</p> <pre>local data = {6000} SetHoldRegs(2048, #data, data, "U16")</pre> <p>Example2: Set a 64-bit double-precision floating-point number starting at address 2048</p> <pre>local data = {95.32105} SetHoldRegs(2048, #data, data, "F64")</pre>

2.16 ECP

ECP (External Control Point) is a coordinate system data used for defining the robot position and orientation at a processing point on the tip of the outside fixed tool, as shown in Figure 2.2. After setting the ECP, the robot can grab a part and move following the specified trajectory around the ECP, for example, sewing application.

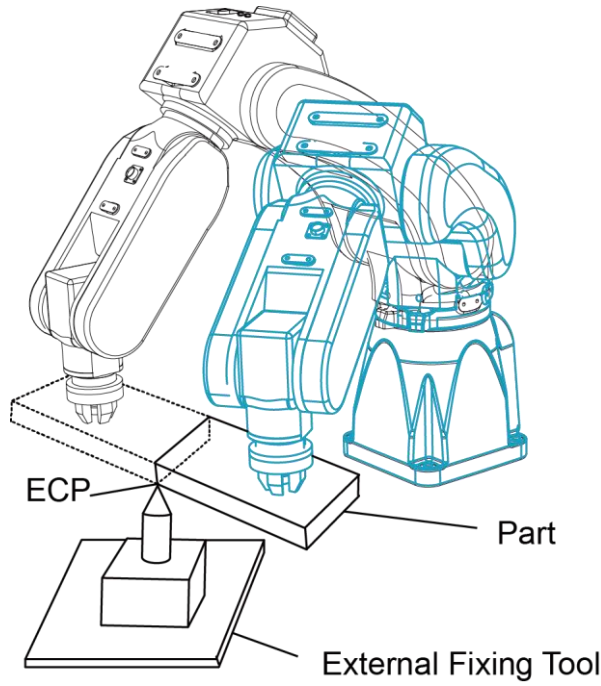


Figure 2.2 ECP

Table 2.58 Enable ECP command

Function	<code>ECP(isOn)</code>
Description	Enable the ECP function
Parameter	isOn: Whether to enable the ECP. ON: Enable; OFF: Disable
Return	None
Example	Please refer to Program 2.5

Table 2.59 Set ECP command

Function	<code>ECPSet(point)</code>
Description	Set ECP This command is valid only when enabling the ECP
Parameter	point: Taught point
Return	None
Example	<code>P1={armOrientation="left", coordinate={630,-32, 95.5,90}, tool=0, user=0}</code> <code>ECPSet(P1)</code>

Program 2.5 ECP demo

```

while true do
    ECPSet(P1)
    ECP(ON)
    Move(P2, "SpeedS=80 CP=10")
    Move(P3, "SpeedS=80 CP=10")
    Arc3(P4,P5, "SpeedS=80 CP=10")
    Move(P6, "SpeedS=80 CP=10")
    ECP(OFF)
end
    
```

2.17 Process Command

2.17.1 Conveyor Tracking Command

Table 2.60 Set conveyor parameter command

Function	<code>CnvVison(<i>CnvID</i>)</code>
Description	Set conveyor number to create a tracing queue
Parameter	CnvID: Conveyor number
Return	0: No error 1: Error
Example	<code>CnvVison(1)</code> Send the information (resolution ratio, Starting position, direction and bound) of Conveyor 1 to the robot system

Table 2.61 Obtain status of the object

Function	<code>GetCnvObject(<i>CnvID</i>, <i>ObjID</i>)</code>
Description	Obtain the information of the part on the conveyor to check whether the part is in the pickup area
Parameter	CnvID: Conveyor index ObjID: Part index
Return	Part status: Whether there is a part. Value range: true or false Part type Part coordinate (x,y,r)

Example	<pre> P111 = {0,0,0} while true do flag,typeObject,P111 = GetCnvObject(0,0) if flag == true then break end Sleep(20) end </pre>
---------	---

Table 2.62 Set offset command

Function	SetCnvPointOffset(xOffset,yOffset)
Description	Set X,Y axes offset under the set User coordinate system
Parameter	xOffset: X axis offset yOffset: Y axis offset Unit: mm
Return	0: No error 1: Error

Table 2.63 Set time compensation command

Function	SetCnvTimeCompensation (time)
Description	Set time compensation This command is used for compensating the pick-up position offset in the moving direction of the conveyor which is caused by taking photos with a time delay
Parameter	time: time-offset. Unit: ms
Return	0: No error 1: Error

Table 2.64 Synchronize conveyor command

Function	SyncCnv (CnvID)
Description	Synchronize the specified conveyor The motion commands used between SyncCnv(CnvID) and StopSyncCnv(CnvID) only support Move command

Parameter	CnvID: Conveyor index
Return	0: No error 1: Error

Table 2.65 Stop synchronizing conveyor command

Function	<code>StopSyncCnv (CnvID)</code>
Description	Stop synchronizing the conveyor The other commands following this command will not be executed until this command running is completed
Parameter	CnvID: Conveyor index
Return	0: No error 1: Error

2.17.2 Pallet Commands

Table 2.66 Create matrix pallet command

Function	<code>Pallet = MatrixPallet (index, "IsUnstack= true Userframe= I")</code>
Description	Instantiate matrix pallet
Parameter	Index: Matrix pallet index Optional parameter: IsUnstack: Stack mode. Value range: true or false. true: Dismantling mode . false: Assembly mode. If not set, the default is assembly mode Userframe: User coordinate system index. If not set, the default is User 0 coordinate system
Return	Matrix pallet object
Example	<code>myPallet = MatrixPallet(0, "IsUnstack=tsrue Userframe=8")</code>

Table 2.67 Set the next stack index command

Function	<code>SetPartIndex (Pallet, index)</code>
Description	Set the next stack index which is to be operated
Parameter	Pallet: Pallet object index: 0 The next stack index. Initial value: 0
Return	None

Example	<pre>local myPallet = MatrixPallet(0, "IsUnstack=true Userframe=8") SetPartIndex(myPallet,1)</pre> <p>The next stack index to be operated is 2</p>
---------	--

Table 2.68 Get the current operated stack index

Function	GetPartIndex (Pallet)
Description	Get the current operated stack index
Parameter	Pallet: Pallet object
Return	The current operated stack index
Example	<pre>local index=GetPartIndex(myPallet)</pre> <p>If the return value is 1, it indicates that the current operated stack index is 2</p>

Table 2.69 Set the next pallet layer index command

Function	SetLayerIndex (Pallet, index)
Description	Set the next pallet layer index which is to be operated
Parameter	Pallet: Pallet object index: The next pallet layer index. Initial value: 0
Return	None
Example	<pre>local myPallet = MatrixPallet(0, "IsUnstack=true Userframe=8") SetPartIndex(myPallet,1)</pre> <p>The next pallet layer index to be operated is 2</p>

Table 2.70 Get the current pallet layer index command

Function	GetLayerIndex (Pallet)
Description	Get the current pallet layer index
Parameter	Pallet: Pallet object
Return	The current pallet layer index
Example	<pre>local index=GetLayerIndex(myPallet)</pre> <p>If the return value is 1, it indicates that the current operated pallet layer index is 2</p>

Table 2.71 Reset command

Function	<code>Reset</code> (Pallet)
Description	Reset pallet
Parameter	Pallet: Pallet object
Return	None
Example	<pre>local myPallet = MatrixPallet(0, "IsUnstack=true Userframe=8") Reset(myPallet)</pre>

Table 2.72 Check the pallet status command

Function	<code>IsDone</code> (Pallet)
Description	Check whether the stack assembly or dismantling is complete
Parameter	Pallet: Pallet object
Return	true: Finished false: Un-finished
Example	<pre>Result = IsDone(myPallet) If (result == true) ... </pre>

Table 2.73 Release pallet command

Function	<code>Release</code> (Pallet)
Description	Release pallet object
Parameter	Pallet: Pallet object
Return	None
Example	<code>Release(myPallet)</code>

Table 2.74 MoveIn command

Function	<code>MoveIn</code> (Pallet, " <code>velAB=20 velBC=30 accAB=20 accBC=10 CP=20 SYNC=1</code> ")
Description	The robot moves from the current position to the first stack position as the configured stack assembly path

Parameter	Required parameter: Pallet: Pallet object Optional parameter: <ul style="list-style-type: none"> • velAB: Velocity rate when the robot moves from the transition point to the preparation point. Value range: 1-100 • velBC: Velocity rate when the robot moves from the preparation point to the first stack point. Value range: 1-100 • accAB: Acceleration rate when the robot moves from the transition point to the preparation point. Value range: 1-100 • accBC: Acceleration rate when the robot moves from the preparation point to the first stack point. Value range: 1-100 • CP: Whether to set continuous path function. Value range: 0- 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Return	None
Example	MoveIn(myPallet, “velAB=90 velBC=50”)

Table 2.75 MoveOut command

Function	MoveOut (Pallet, “velAB=20 velBC=30 accAB=20 accBC=10 CP=20 SYNC=1”)
Description	The robot moves from the current position to the transition point as the configured stack dismantling path

Parameter	Required parameter Pallet: Pallet object Optional parameter <ul style="list-style-type: none"> • velAB: Velocity rate when the robot moves from the preparation point to the transition point. Value range: 1-100 • velBC: Velocity rate when the robot moves from the first stack point to the preparation point. Value range: 1-100 • accAB: Acceleration rate when the robot moves from the preparation point to the transition point. Value range: 1-100 • accBC: Acceleration rate when the robot moves from the first stack point to the preparation point. Value range: 1-100 • CP: Whether to set continuous path function. Value range: 0- 100 • SYNC: Synchronization flag. Value range: 0 or 1. If SYNC is 0, it indicates asynchronous execution, this command has a return immediately after calling it, regardless of the command process. If SYNC is 1, it indicates synchronous execution. After calling this command, it will not return until it is executed completely
Return	None
Example	MoveOut(myPallet, "velAB=90 velBC=50")

 NOTE

Figure 2.3 and Figure 2.4 show the stack assembly path and dismantling path respectively. Point A is the transition point, which is fixed or varies with the pallet layer. Point B is the preparation point which is calculated by the target point and the set offset. Point C is the first stack point.

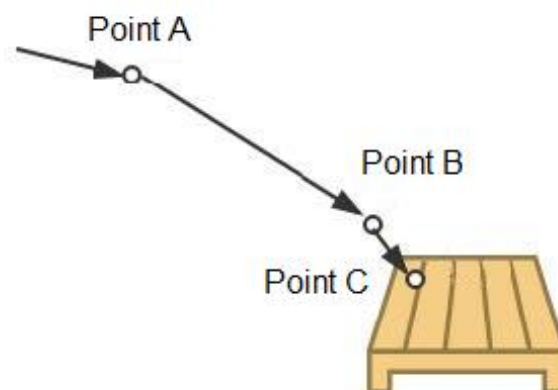


Figure 2.3 Stack assembly path

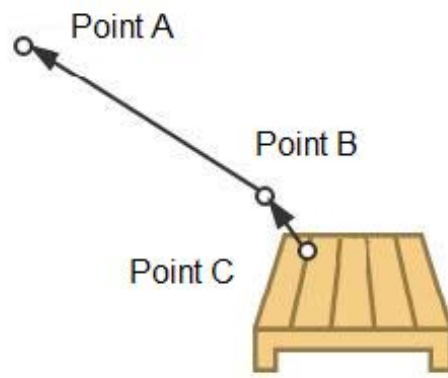


Figure 2.4 Stack dismantling path

3. Process Guide

3.1 Conveyor Tracking

3.1.1 Overview

Conveyor tracking is that the vision system or sensor system finds the parts on the conveyor when conveyor moves constantly and the robot picks them up as they move.

3.1.2 Building Environment

Figure 3.1 shows the communication process of conveyor tracking. Vision system or photoelectric sensor for detection is selected based on site requirements. If the photoelectric sensor is used, the part is detected by a change in the digital input data. If the vision system is used, the part is detected by the camera and this is triggered by the rising edge of digital output signal.

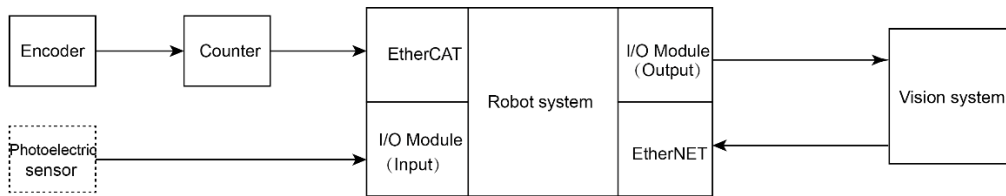


Figure 3.1 Communication process of the conveyor tracking

Figure 3.2 shows the full process environment of conveyor tracking. Please select a vision system or photoelectric sensor for detection based on site requirements.

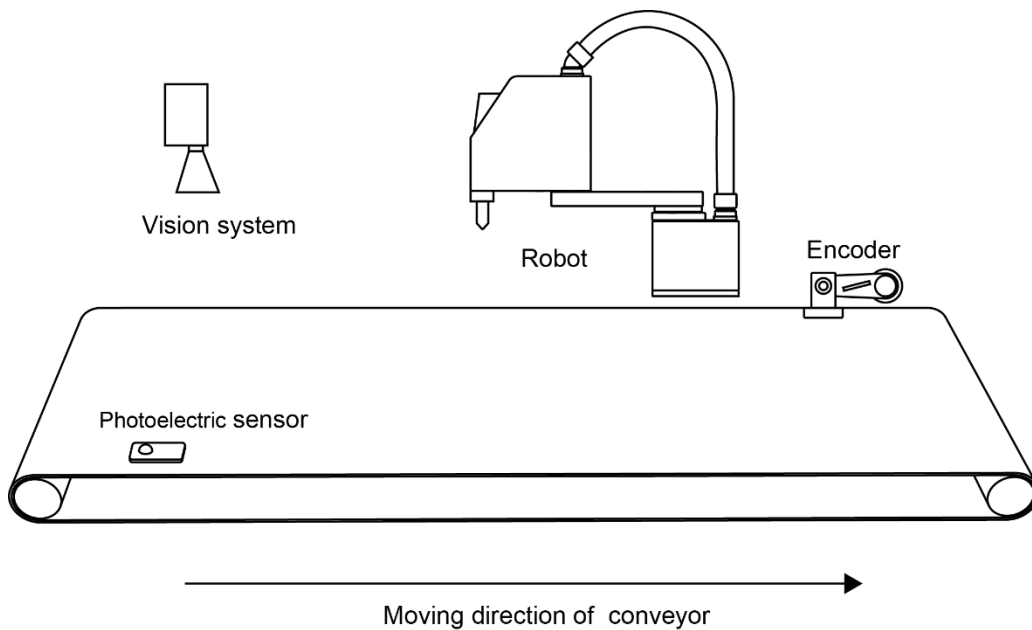


Figure 3.2 Process environment of the conveyor tracking

3.1.2.1 Encoder

An Encoder is used for recording the conveyor moving distance and the part position and reporting them to the robot system by a counter. Please connect the Encoder to the **EtherCAT** interface on the robot system with the high-speed counter and communication module (called them as counter module), as shown in Figure 3.3. The E6B2-CWZ1X(1000P/R)Encoder, Beckhoff EL5101 high-speed counter, and Beckhoff EK1100 communication module are recommended in this application.

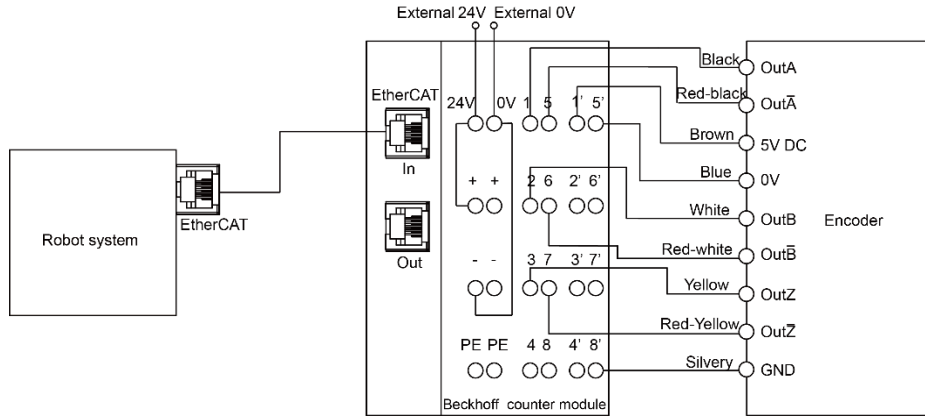


Figure 3.3 Connection between Encoder and robot system

3.1.2.2 Photoelectric Sensor

The photoelectric sensor outputs different level signals according to whether the part is detected or not. When you connect the photoelectric sensor to a DI interface on the robot system, the photoelectric sensor can detect parts by a change in this DI interface.

3.1.2.3 Vision System

A vision system is communicated with the robot system by the TCP/IP protocol and is triggered by the DO interface on the robot system to detect the part.

- If the robot system version is earlier than **V3.0.0.20190219121343**, the **DO 16** is triggered to take photos for detecting, as shown in Figure 3.4. You can check the robot system version on the **Config>BasicConfig>Version** page of the DobotSCStudio.

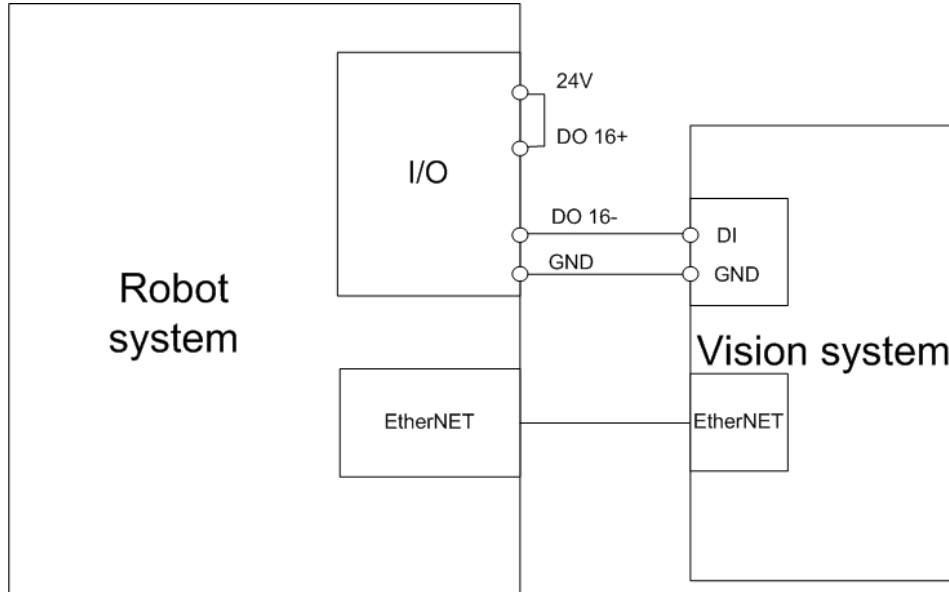


Figure 3.4 Vision system connection (1)

- If the robot system version is **V3.0.0.20190219121343** or later, the **DO 10** is triggered to take photos for detecting, as shown in Figure 3.5.

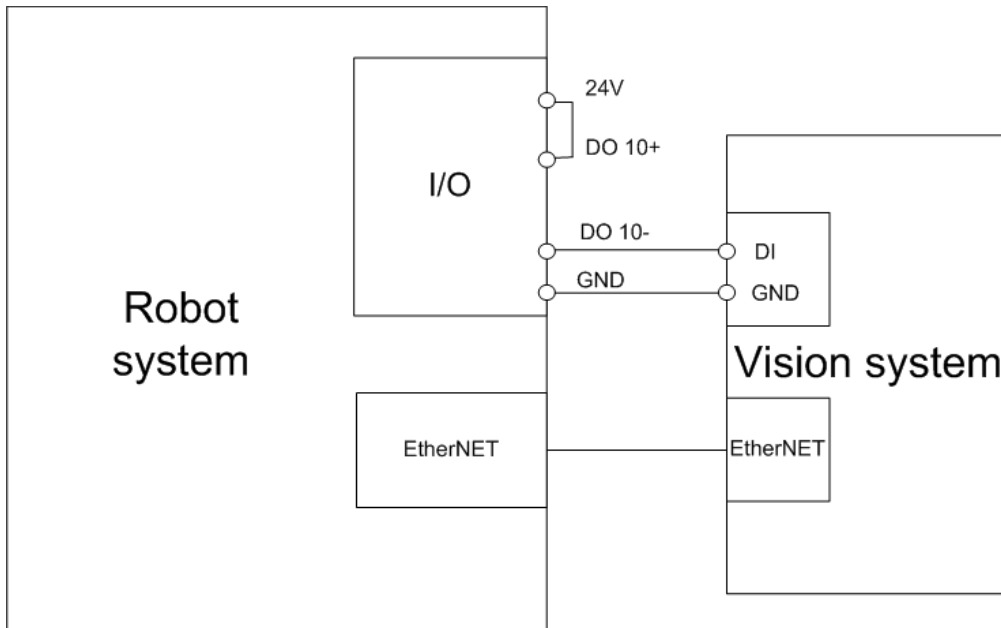


Figure 3.5 Vision system connection (2)

3.1.3 Calibrating Conveyor

Before tracking parts, please calibrating the conveyor, for obtaining the positional relationship between the conveyor and the robot. In the following steps, we assume that the Y-axis positive

direction under the base coordinate system is coincident with the moving direction of the conveyor.

Prerequisites

- The robot has been powered on.
- The calibration kit has been installed at the end of the robot.

Procedure

Step 1 Put down a label on the conveyor, as shown in Figure 3.6.

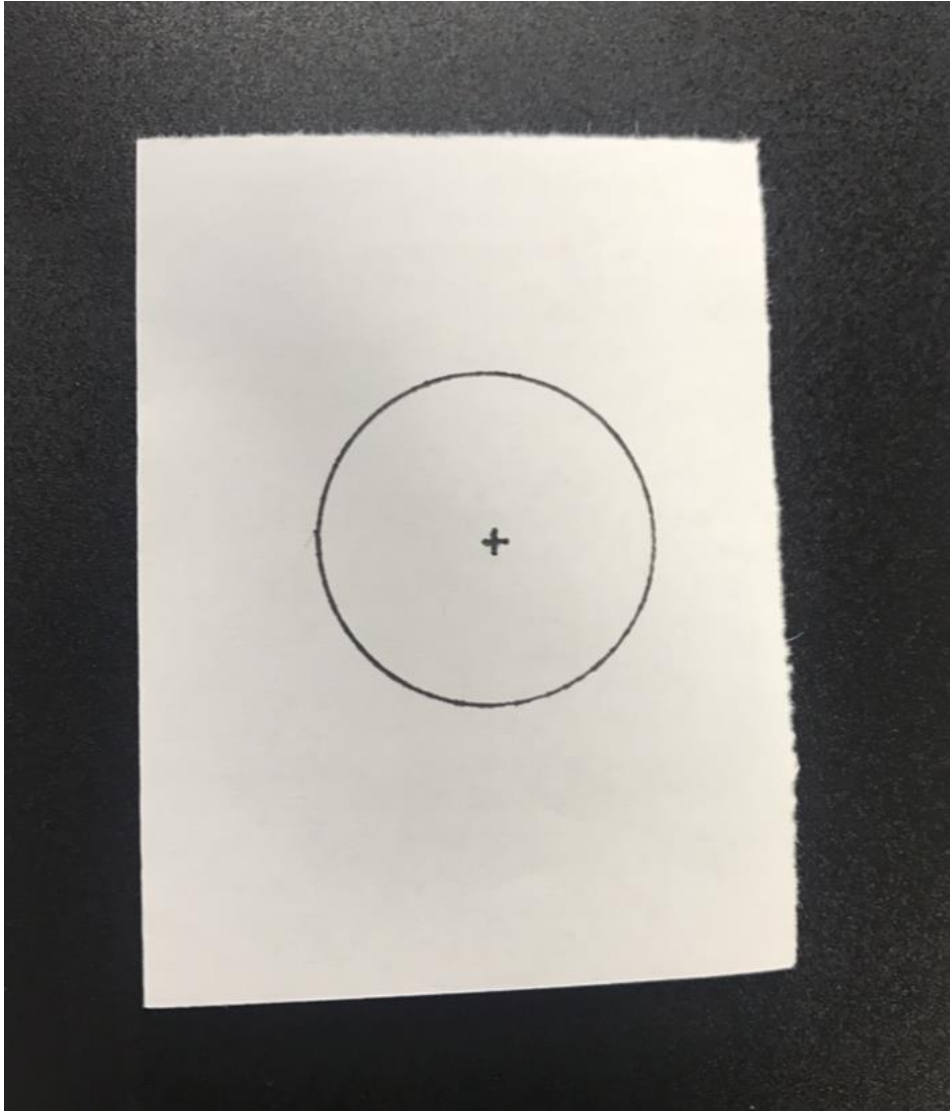


Figure 3.6 Put down a label on the conveyor

Step 2 Click  > **Config** > **GlobalCoordinate** > **Coordinate User**.

The **Coordinate User** page is displayed, as shown in Figure 3.7.

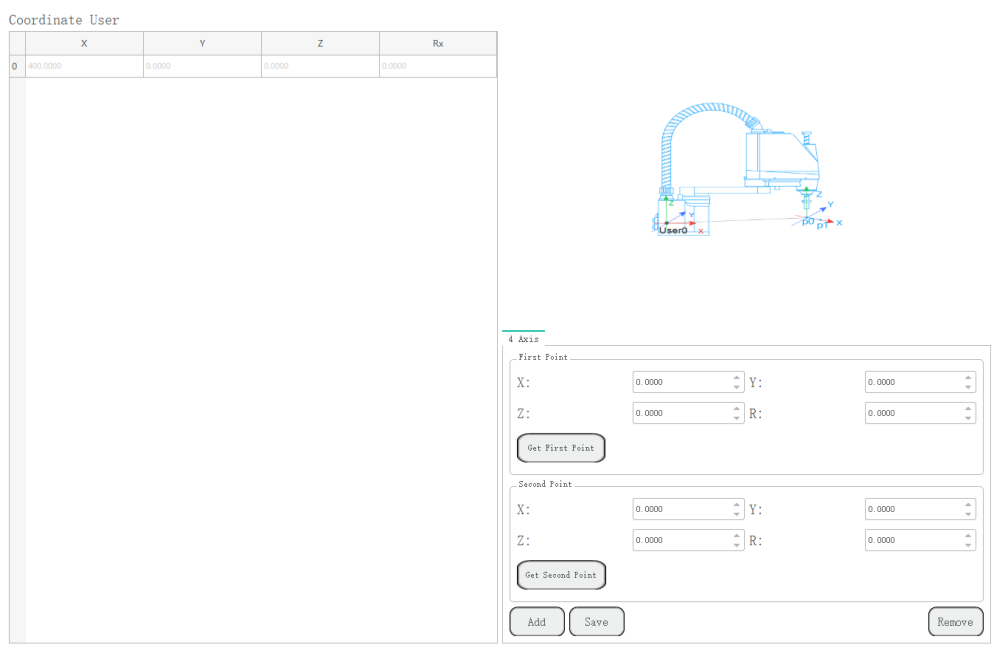


Figure 3.7 User coordinate system page

Step 3 Enable the motor and jog the robot to the label position on the conveyor, and click **Get First Point** on the **First Point** section, as shown in Figure 3.8. We called this point as point A, which is the origin of the User coordinate system.

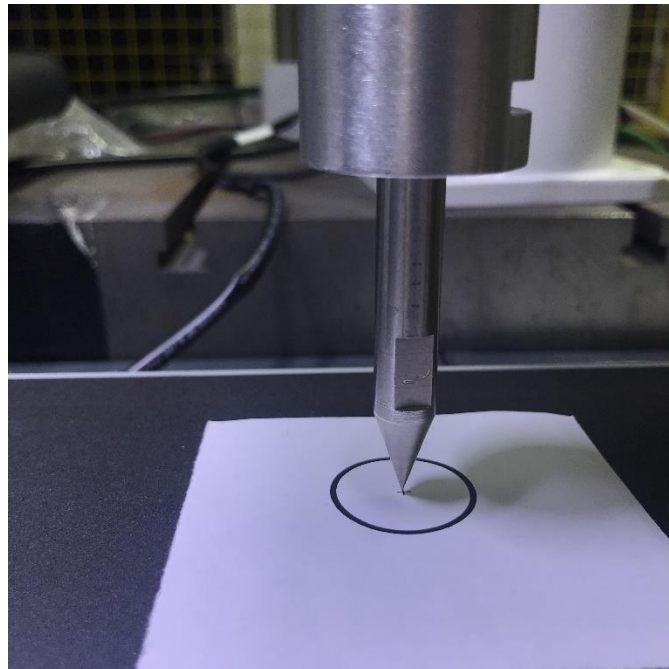


Figure 3.8 Conveyor calibration

Step 4 Control the conveyor move a specified distance.

- Step 5** Enable the motor and jog the robot to the label position on the conveyor, and click **Get Second Point** on the **Second Point** section. We called this point as point B. The line from point A to point B is defined as the positive direction of X-axis. And then the Y-axis and Z-axis can be defined based on the right-handed rule, as shown in Figure 3.9.

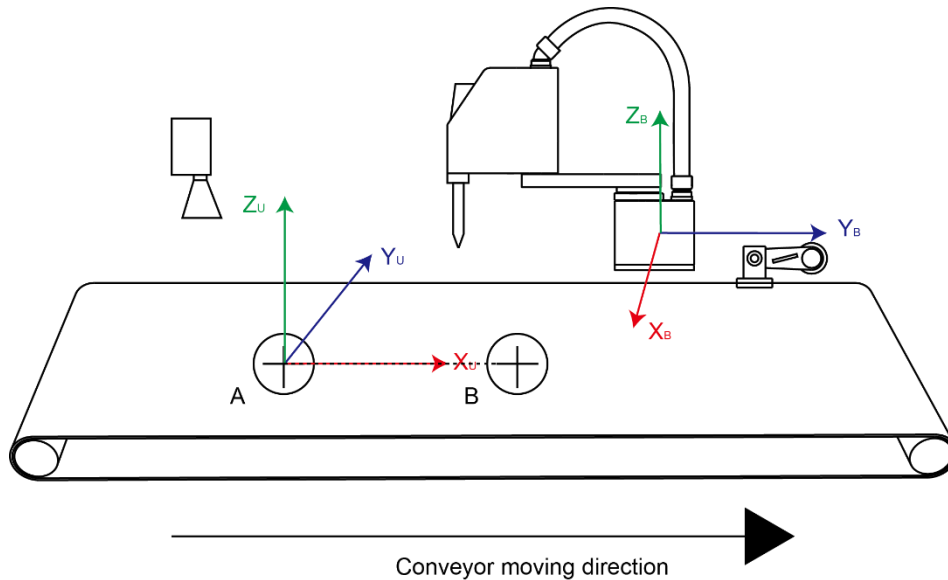


Figure 3.9 User coordinate system calibration

- Step 6** Click **Add** to generate the User coordinate system.

We assume that the User 1 coordinate system is used. If the R-axis coordinate is 90° , it indicates that the calibration is successful. Otherwise, please re-calibrate it.

⚠ NOTICE

The R-axis coordinate after calibration depends on the positional relationship between robot and conveyor.

- If the moving direction of the conveyor is the same with the X-axis positive direction under the base coordinate system, the R-axis coordinate after calibration is 0° .
- If the moving direction of the conveyor is the same with the X-axis negative direction under the base coordinate system, the R-axis coordinate after calibration is 180° or -180° .
- If the moving direction of the conveyor is the same with the Y-axis positive direction under the base coordinate system, the R-axis coordinate after calibration is 90° .
- If the moving direction of the conveyor is the same with the Y-axis negative direction under the base coordinate system, the R-axis coordinate after calibration is -90° .

3.1.4 Configuring Conveyor

Prerequisites

- The robot has been powered on.
- The calibration kit has been installed at the end of the robot.

NOTICE

- The full process should be operated under the base coordinate system and the matched calibration kit is required.
- Please be sure to follow the steps to operate, otherwise, the parameter setting will fail.

Procedure

Step 1 Click **Process>Tracking**.

The conveyor tracking page is displayed, as shown in Figure 3.10.

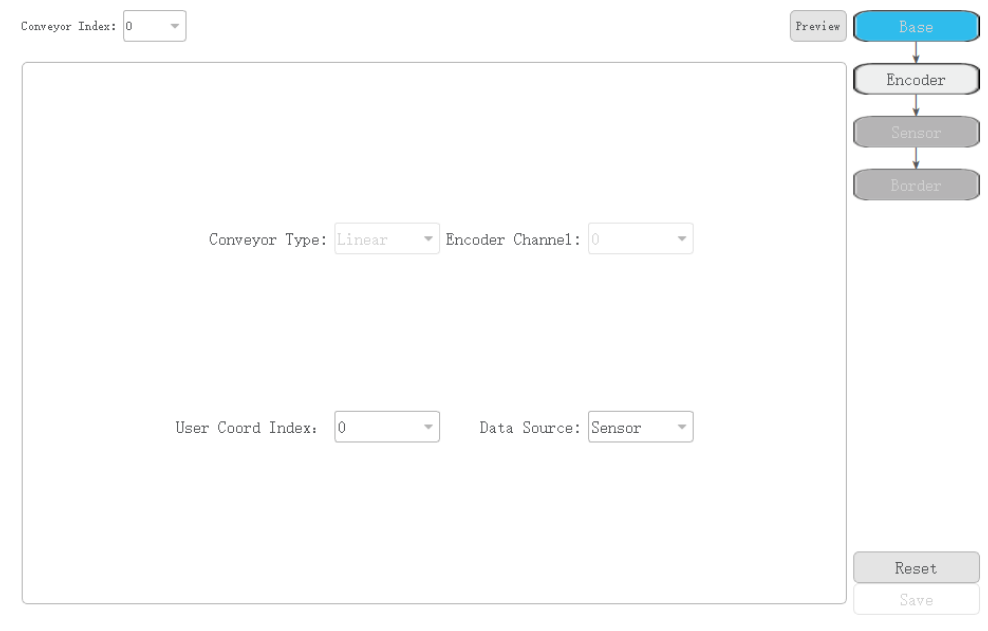


Figure 3.10 Conveyor tracking page

Step 2 Set the basic parameters.

Table 3.1 shows the basic parameter description.

Table 3.1 Basic parameter description

Parameter	Description
Conveyor index	Conveyor index

Parameter	Description
	This parameter cannot be set
Conveyor Type	Conveyor type <ul style="list-style-type: none"> • Linear • Circular Only linear type is supported
Encoder Channel	Encoder channel This parameter cannot be set
User Coord Index	User coordinate system index Please select the right index according to the user coordinate system set in <i>3.1.3 Calibrating Conveyor</i>
Data Source	Detection Mode <ul style="list-style-type: none"> • Sensor: Use a sensor to detect parts • Camera: Use a vision system to detect parts Please select the mode based on site requirements

Step 3 Click **Encoder** to calibrate the Encoder.

1. Put down a label on the conveyor, as shown in Figure 3.11.

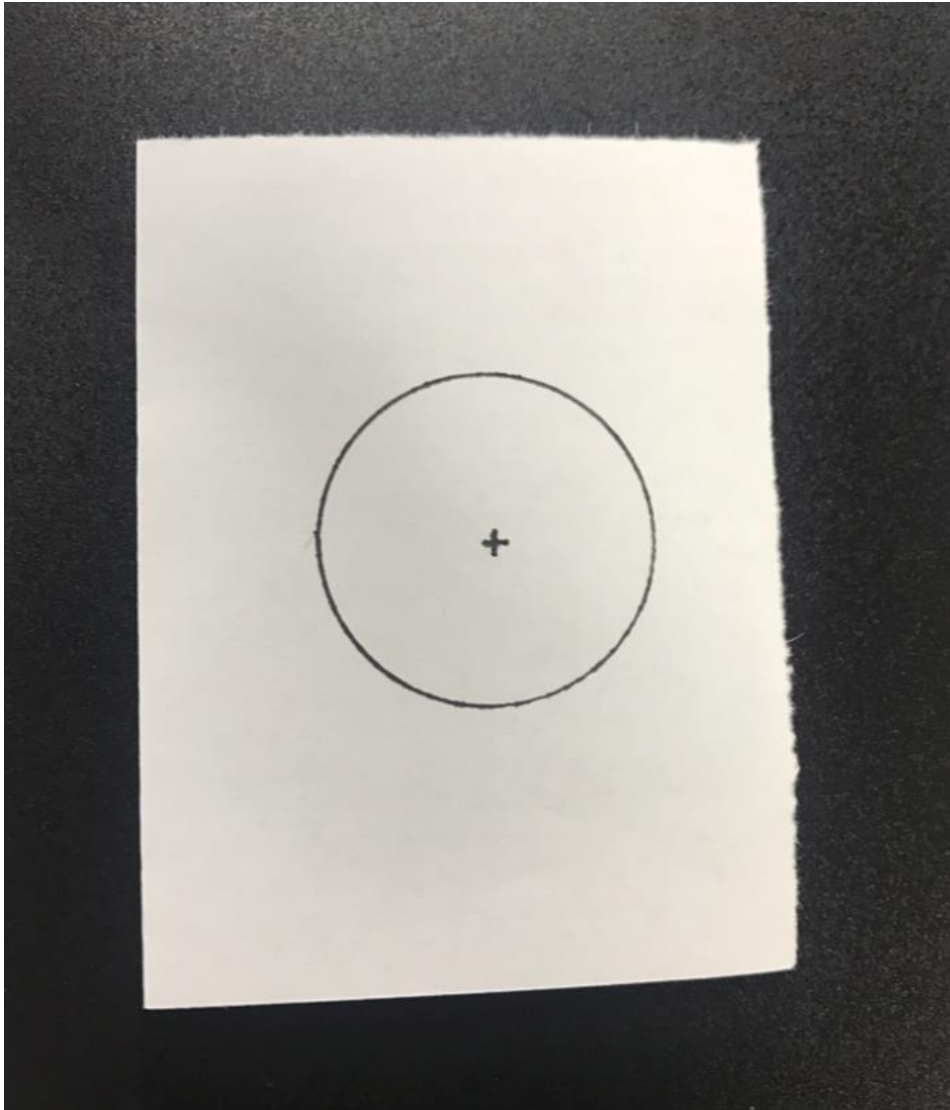


Figure 3.11 Put down a label on the conveyor

2. Enable the motor and jog the robot to the label position on the conveyor, then click **1**, as shown in Figure 3.12 and Figure 3.13.

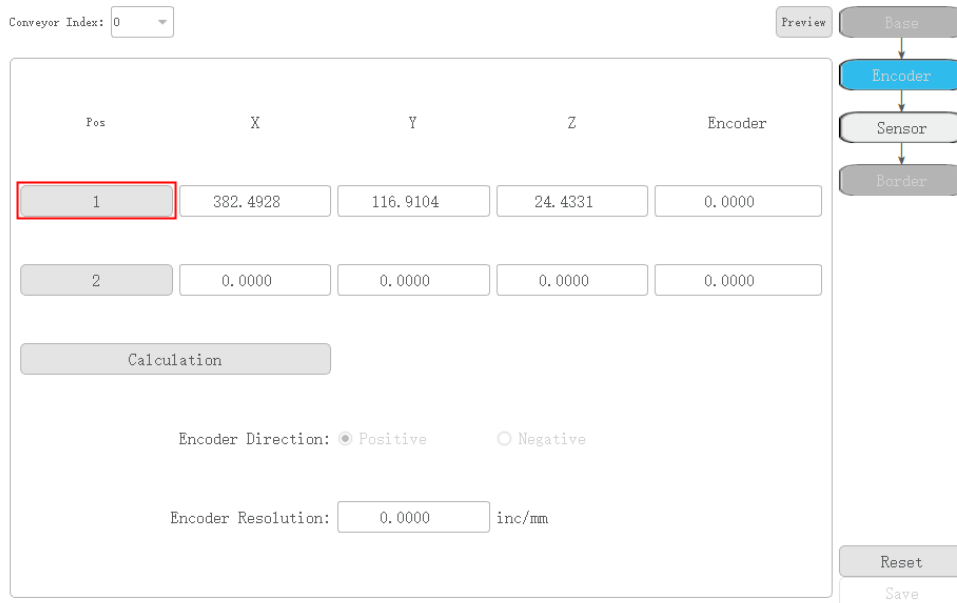


Figure 3.12 Encoder calibration page

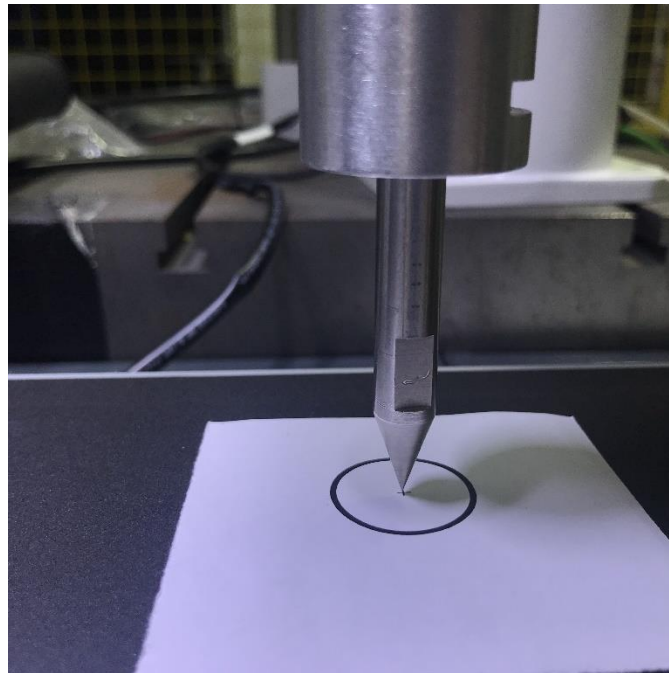


Figure 3.13 Encoder calibration

3. Control the conveyor to move a specified distance.
4. Enable the motor and jog the robot to the label position on the conveyor, then click **2**.
5. Click **Calculation** to obtain the Encoder resolution.

Encoder resolution is the pulse increment of the Encoder per unit length that conveyor moves

NOTE

If Data Source is sensor, please execute **Step 4**. If not, please execute **Step 5**.

Step 4 (Optional) Click **Sensor** to calibrate the sensor, as shown in Figure 3.14.

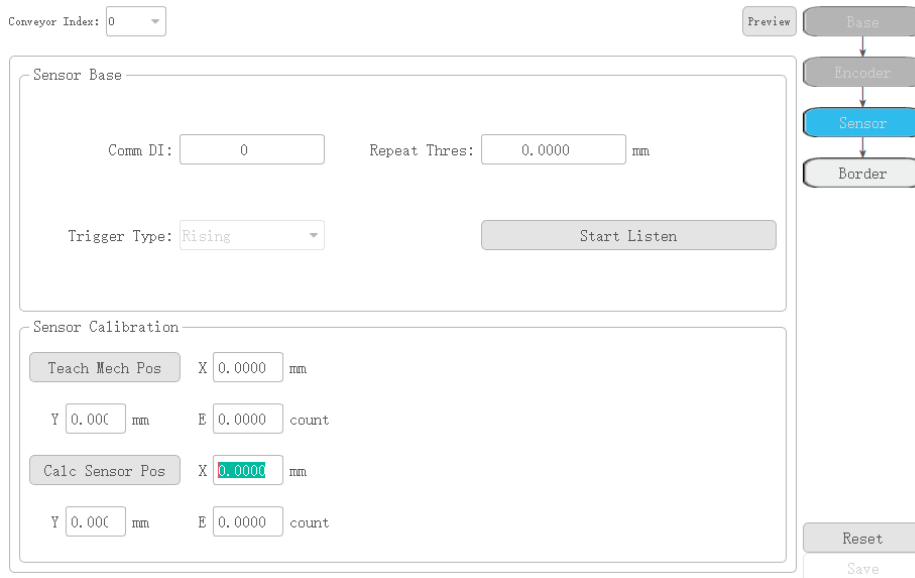


Figure 3.14 Sensor calibration page

Sensor calibration is to obtain the position where the sensor finds the part so that the position of the part under the User coordinate system at every moment can be calculated based on the coordinate offset when the part moves along with the conveyor.

Control the conveyor move to a position where the part on the conveyor is within the workspace of the robot and has been passed the sensor, and jog the robot to the part center for obtaining the current taught position. At the same time, the robot records the moving distance of the conveyor after the part is passed the sensor. According to the current taught position and the moving distance of the conveyor, we can get the part position when the sensor locates the part. For details, please see as follows.

1. Set the related parameters on the sensor setting page.

Table 3.2 lists the sensor parameter description.

Table 3.2 Sensor parameter description

Parameter	Description
Conveyor index	Conveyor index

Parameter	Description
	This parameter cannot be set
Comm DI	This parameter cannot be set
Trigger Type	Signal outputted when the sensor finds the part <ul style="list-style-type: none"> • Raising edge • Falling edge This parameter cannot be set
Repeat Thres	Reject distance. This parameter is set based on site requirements This is used to prevent the registration of the duplicate parts

2. Click **Start Listen**.
3. Put a part on the upstream area of the conveyor and Control the conveyor move. After the sensor finds the part and meanwhile the part is in the workspace of the robot, please control the conveyor stop.
4. Make the 3-position in the middle gear to enable the motor and jog the robot to the part center, then click **Teach Mech Pose** to obtain the current position.
5. Click **Calc Sensor Pos** to obtain the position where the sensor finds the part

Step 5 (Optional) Click **Camera** to calibrate the vision system, as shown in Figure 3.15.

Before calibrating the vision system, you need to set the robot IP address and port on the vision software for communication between the robot and vision system with the TCP/IP protocol. The robot IP address is 192.168.5.1, and the port is 8080. The data format is as follows.

- The parts found in the same frame are sent together.
- The data format of a part is (x,y,r,classID). classID is the part type.
- Different part data is separated by a semicolon.
- A frame is terminated by & character.

e.g.: x1,y1,r1,1;x2,y2,r2,4;x3,y3,r3,2&

NOTICE

Vision software depends on the vision brand. So, the settings about the vision system are different. The details on how to use the vision software are not described in this topic.

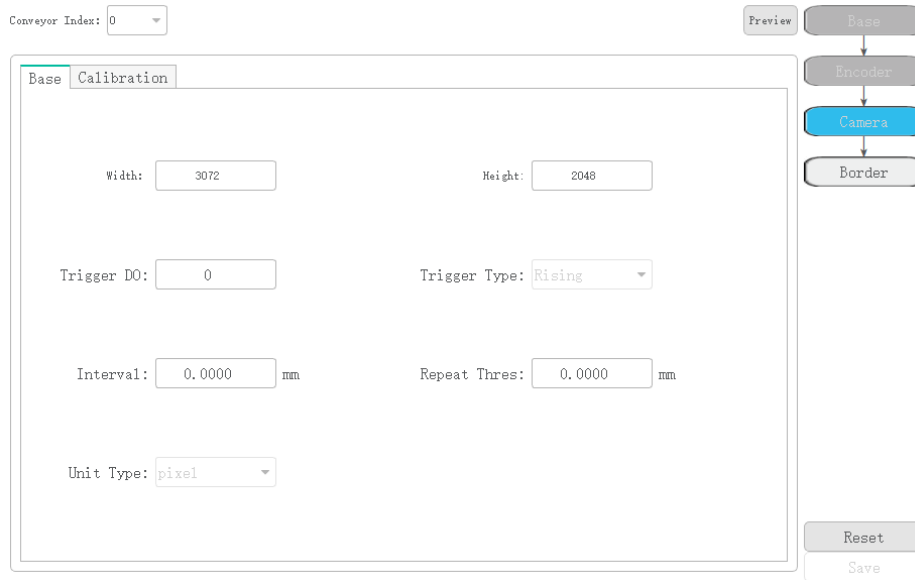


Figure 3.15 Vision system calibration page

The vision system is to obtain the part coordinate under the User coordinate system where the camera finds it from the conveyor. So that the position of the part under the User coordinate system at every moment can be calculated based on the coordinate offset when the part moves along with the conveyor.

Place the calibration board on the conveyor which is in the vision search area and obtain the image coordinates of the nine points on the calibration board and record the current position of the conveyor. Move the conveyor to a position where the calibration board is in the robot workspace and jog the robot to the nine points on the calibration board and obtain their Cartesian coordinates respectively. At the same time, record the current position of the conveyor. Based on the Cartesian coordinates and image coordinates of the nine points and the moving distance of the conveyor, we can calculate the Cartesian coordinates where camera finds the calibration board and obtain the relationship between the image coordinate and the Cartesian coordinate. For details, please see as follows.

1. Set the basic parameters of the vision system on the **Base** tab.

Table 3.3 lists the basic parameter description.

Table 3.3 Basic parameter description

Parameter	Description
Conveyor Index	Conveyor index This parameter cannot be set
Width	Resolution width

Parameter	Description
	Default value: 3072
Height	Resolution height Default value:2048
Trigger DO	this parameter cannot be set
Trigger Type	This parameter cannot be set
Interval	Interval photography Namely, the robot gives a DO signal to trigger photography each time conveyor moves a given distance The recommended value range is $(1-d)/2 < \text{Interval} < (1-d)$ l indicates the width of the vision search area in the conveyor moving direction and d indicates the maximum width of the part
Repeat Thres	Reject distance. This parameter is set based on site requirements This is used to prevent the registration of the duplicate parts
Unit Type	This parameter cannot be set

2. Click **Calibration**.

The calibration page is displayed.

3. Place the calibration board on the conveyor which is in the vision search area and click **Conveyor pose** to record the current Encoder value. The calibration board is shown in Figure 3.16.

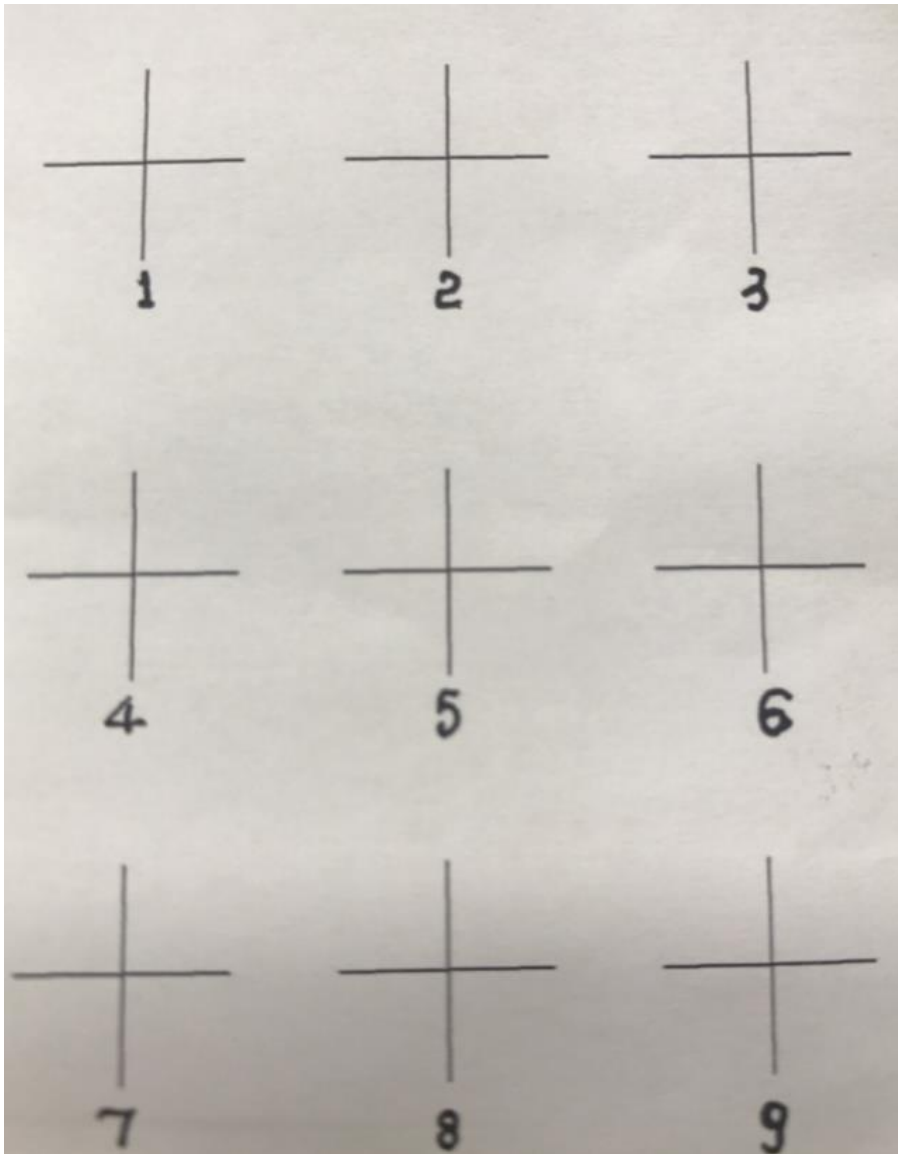


Figure 3.16 Calibration board

4. Get the image coordinates of the nine points on the calibration board from the vision software after the vision system finds the calibration board and input them to the corresponding positions on the calibration page, as shown in Figure 3.17.

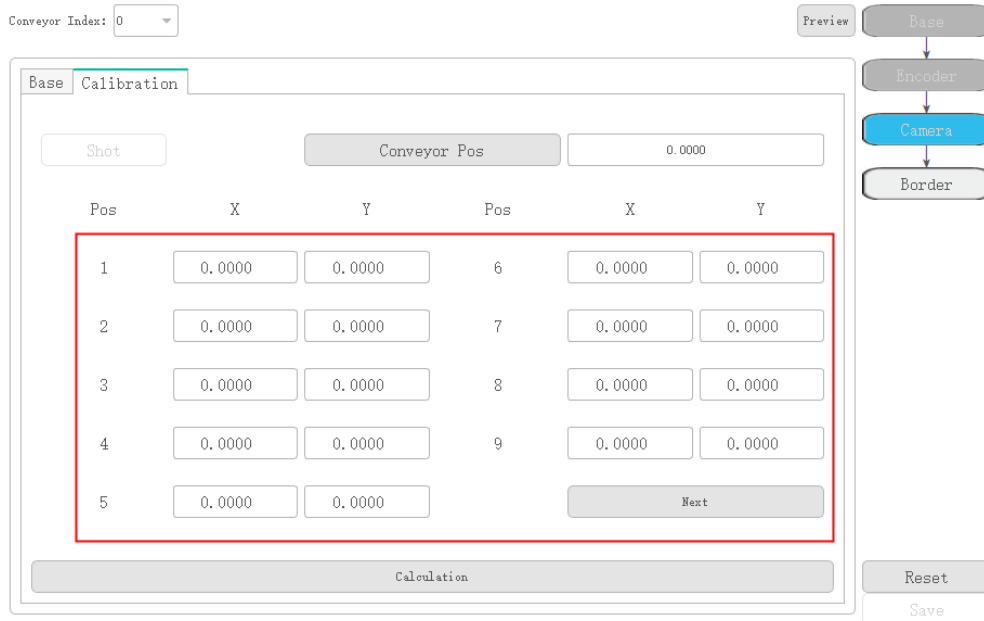


Figure 3.17 Get the image coordinates

5. Click **Next**.
6. Move the conveyor to the position where the calibration board is in the robot workspace and stop, then click **Conveyor Pos** to obtain the current Encoder value.
7. Enable the motor and jog the robot to the corresponding nine points on the calibration board and click the right index on the calibration page respectively as shown in Figure 3.18 and Figure 3.19.

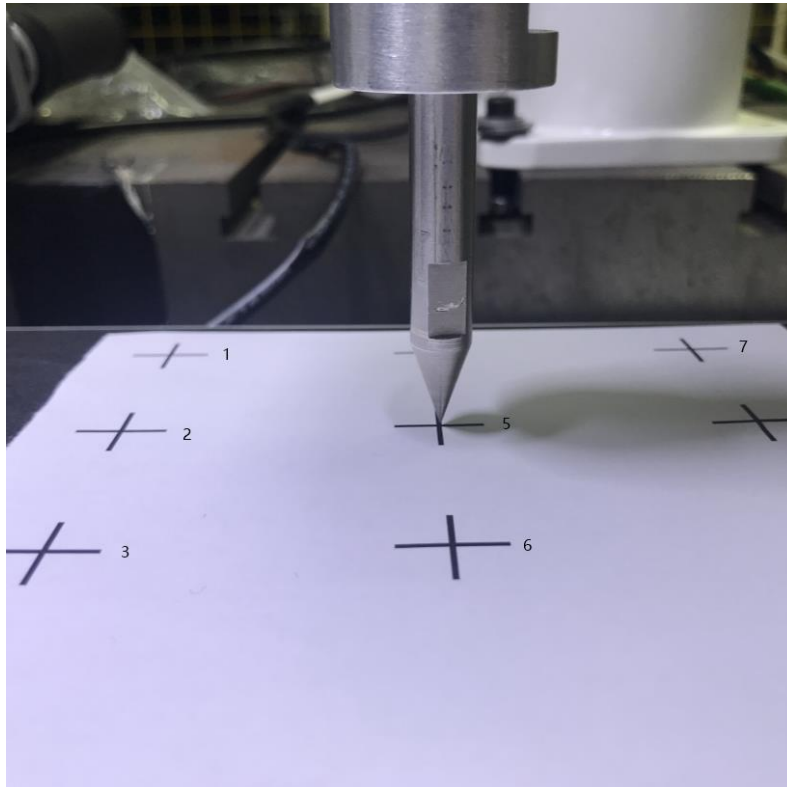


Figure 3.18 Vision calibration

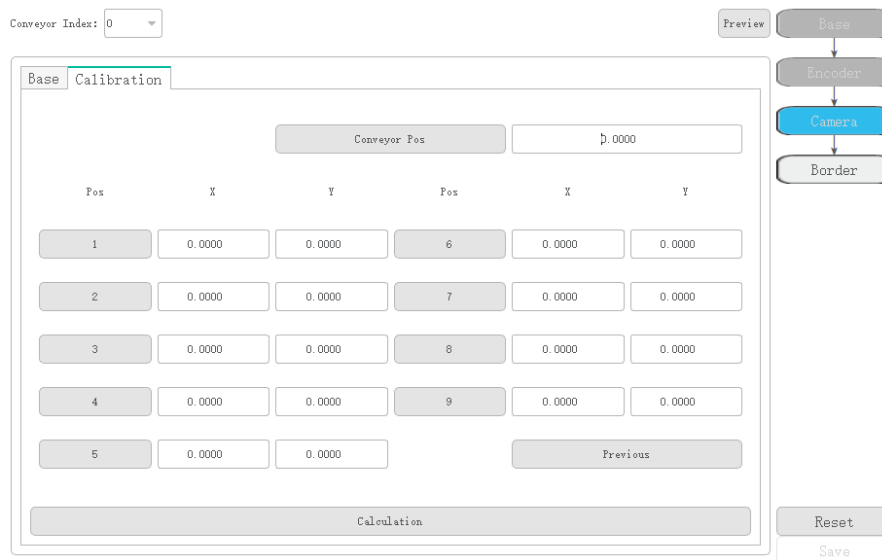


Figure 3.19 Get the Cartesian coordinates of the nine points on the calibration board

NOTICE

Please be sure to obtain the Cartesian coordinates in the order of the image coordinates of the nine points to avoid vision calibration errors.

8. Click **Calculation** to calculate the Cartesian coordinates of the nine points as the vision system finds them and obtain the relationship between the image coordinate and the Cartesian coordinate based on the image coordinates and the Cartesian coordinates of the nine points and the moving distance of the conveyor.

Step 6 Click **Border** to calibrate the border.

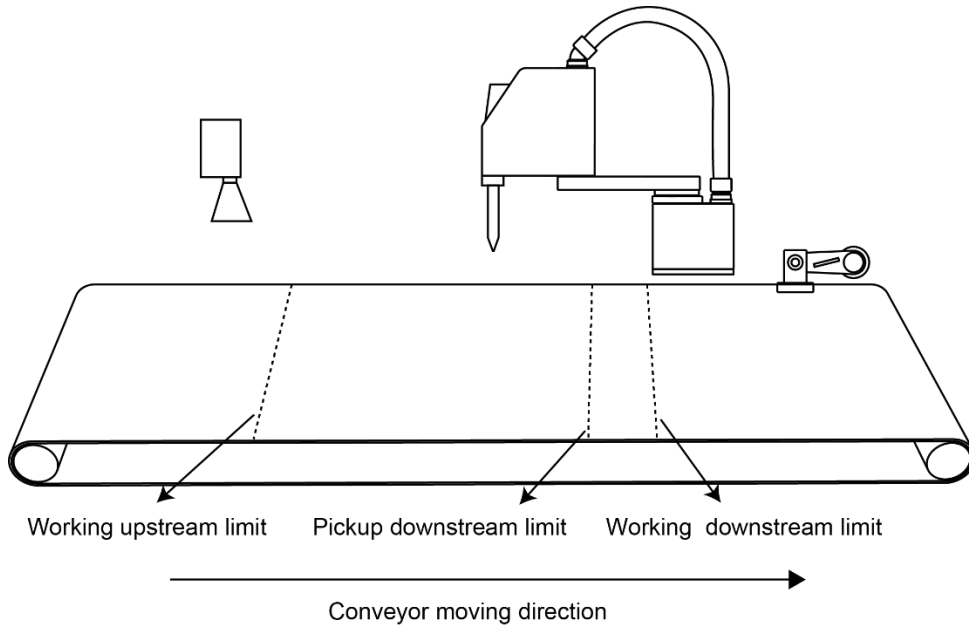


Figure 3.20 Border description

Conveyor Index:

Preview

Base

Encoder

Sensor

Border

Reset

Save

Working upstream limit
Pickup downstream limit
Working downstream limit

Working upstream limit	<input style="width: 90%;" type="text" value="0.0000"/>	mm
Pickup downstream limit	<input style="width: 90%;" type="text" value="0.0000"/>	mm
Working downstream limit	<input style="width: 90%;" type="text" value="0.0000"/>	mm
Stop Distance	<input style="width: 90%;" type="text" value="0.0000"/>	mm

Figure 3.21 Border calibration page

Table 3.4 lists the border parameter description

Table 3.4 Border parameter description

Parameter	Description
Conveyor Index	Conveyor Index This parameter cannot be set
Working upstream limit	Working upstream limit
Pickup downstream limit	Pickup downstream limit When the part moves out of the Pickup downstream limit, the robot will not pick up it.
Working downstream limit	Working downstream limit
Stop distance	This parameter cannot be set

1. Enable the motor and jog the robot to the starting tracking position, then click **Working upstream limit** to obtain the working upstream limit.
2. Enable the motor and jog the robot to the latest starting tracking position that is expected to complete the process, then click **Working upstream limit** to obtain the working upstream limit.
Please set this parameter according to the conveyor speed and practical experience.
3. Enable the motor and jog the robot to the end tracking position, then click **Working downstream limit** to obtain the working downstream limit.

Step 7 Click **Save**.

3.1.5 Example

3.1.5.1 Pickup Example Using Vision Conveyor Tracking

In this application, we need to teach six points.

- Waiting point: P1
- Tracking point: P2
- Pickup point: P3
- Lifting point: P4
- Point above the placing point: P5
- Placing point: P6

If the pickup angle is required, you need to set this angle.

This topic takes the pickup application without angle requirement as an example, Figure 3.22 shows the taught positions.

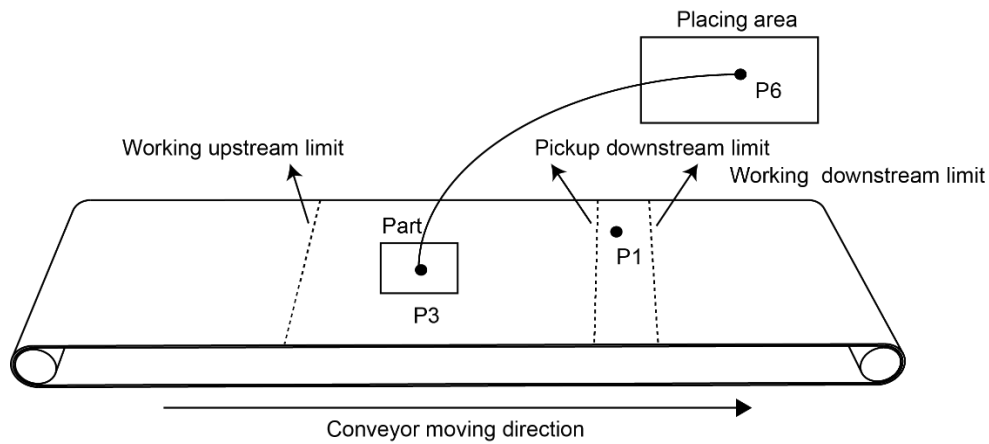


Figure 3.22 Taught position

Prerequisites

- The vision conveyor tracking project has been created on DobotSCStudio. For details, please see *1.5.3.1 Creating Project*.
- The robot has been connected to the air pump.
- The end effector has been mounted on the end of the robot.
- The robot has been switched to the manual mode.
- The vision software has been installed.
- (Optional) If an eccentric end effector has been mounted, please set the Tool coordinate system. For details, please see *1.2.3.1 Setting Tool Coordinate System of SCARA Robot*.

NOTICE

- Enable the robot motor when jogging the robot.
- Vision software depends on the vision brand. The details on how to set and create a template are not described in this topic.
- The arm orientations of P1, P2, P3, and P4 must be the same.
- You need to teach P2, P3 and P4 points under the set User coordinate system. If an eccentric end effector is used, you need to set the Tool coordinate system and then teach P2, P3, P4 points under the set Tool coordinate system.
- The J3 angle of P1 point is recommended to set to 120°, and the J4 angle is recommended to set to 0°.

Procedure

- Step 1** Jog the robot to the point P1 under the basic coordinate system and click on the **TeachPoint** page to add the saved point information.

NOTICE

If the eccentric end effector is mounted, please execute **Step 2 - Step 3** under the set Tool

coordinate system.

- Step 2** Jog the robot to a right height (which is higher than the part) under the set User coordinate system and record the height of the point P2 which is called **z1**.
- Step 3** Jog the robot to the center of the part under the set User coordinate system and record the height of the point P3 which is called **z0**.
- Step 4** Add P2, P3 and P4 points on the **TeachPoint** page under the basic coordinate system. P2=(0,0,z1,0), P3=(0,0,z0,0), P4=(0,0,z1,0)
- Step 5** Put a part on the conveyor which is in the vision search area and create the template on the vision software to extract the part features.

Please set the current template angle to 0° .

 **NOTE**

If you use the sensor to detect parts, please place the parts in the correct direction (The parts detected by the sensor on the conveyor have a fixed direction)


- Step 6** (Optional) This step is executed only when the pickup angle is required. If there is no requirement, please skip this step.


1. Move the conveyor to a position where the part is in the pickup area, jog the robot to the center of this part, and set the R-axis to 0° under the basic coordinate system.
2. Switch to the set User coordinate system and record the R-axis value which is called **r1**.
3. Adjust the R-axis to a right pose to pick up the part, and record the R-axis value which is called **r2**.

Therefore, the R-axis value **r** of P2, P3 and P4 points is **r2 – r1**.

- $r > 180^\circ$, $r = r - 360^\circ$
- $r < -180^\circ$, $r = r + 360^\circ$
- $-180 \leq r \leq 180^\circ$, **r** remains unchanged

4. Modify the P2, P3 and P4 points on the **TeachPoint** page. P2=(0,0,z1,r), P3=(0,0,z0,r), P4=(0,0,z1,r)

- Step 7** Jog the robot to point P6 under the basic coordinate system, and click  on the **TeachPoint** page to add the saved point information.

- Step 8** Jog the robot to point P5 under the basic coordinate system, and click  on the **TeachPoint** page to add the saved point information.

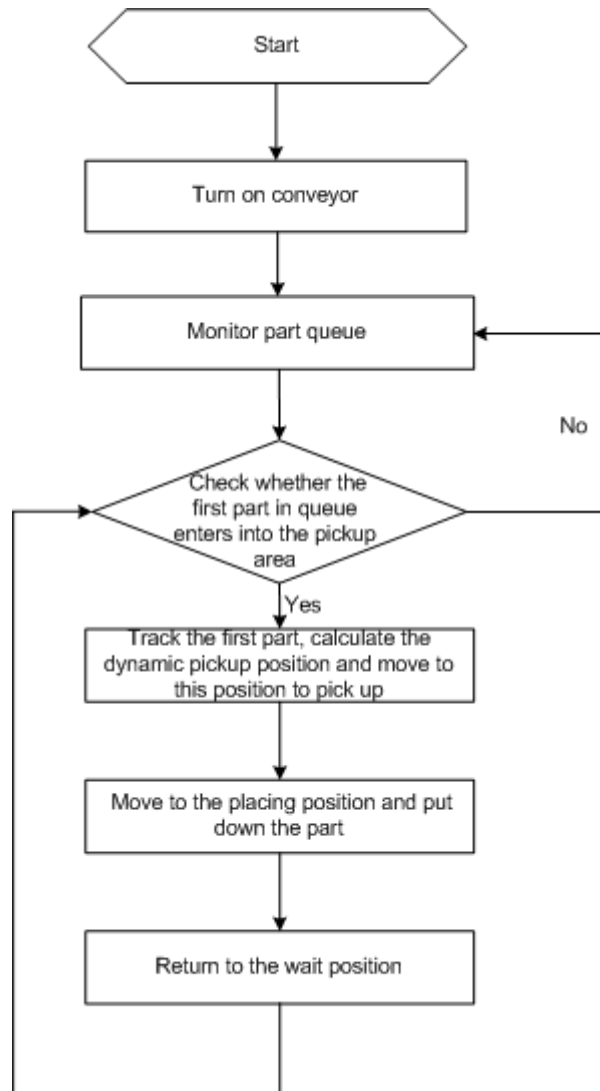


Figure 3.23 Conveyor tracking process

 NOTICE

- The motion commands used between SyncCnv(0) and StopSyncCnv() only support Move command
- The wait, DI and DO commands are supported between SyncCnv(0) and StopSyncCnv().

Program 3.1 Conveyor tracking program

```

CnvVison(0)           // Activate conveyor
DO(1,0)               // Control the air pump status by DO1 and DO2
DO(2,0)
local flag            //Part flag
    
```

```

local typeObject          /// Part type
local point = {0,0,0}     //Part coordinate (X,Y,R)
while true do
    Go(P1,"Speed=100 Accel=100 SYNC=1")    // Wait point
    print("Test")
    while true do
        flag,typeObject,point = GetCnvObject(0,0) //Check whether there is a part. If there is a part, exit this
loop
        if flag == true then
            break
        end
        Sleep(20)
    end
    SyncCnv(0)                //Synchronize the conveyor and start to track
    Move(P2,"SpeedS=100 AccelS=100")    // Tracking point
    Move(P3,"SpeedS=100 AccelS=100")    // Pickup point
    Wait(100)
    DO(1,1)                    // Active air pump and pick up part
    DO(2,1)
    Wait(100)
    Move(P4,"SpeedS=100 AccelS=100 SYNC=1") // Lifting point
    StopSyncCnv()             // Stop conveyor tracking
    Sleep(20)
    Go(P5,"Speed=100 Accel=100")    // Point above the placing point
    Go(P6,"Speed=100 Accel=100 SYNC=1") // Placing point
    Sleep(1000)
    DO(1,0)                    // Close air pump
    DO(2,0,"SYNC=1")
    Sleep(1000)
    Go(P5,"SpeedS=100 AccelS=100")
end
    
```

3.2 Palletizing

3.2.1 Overview

In carrying applications, some parts are regularly arranged with uniform spacing and teaching part positions one by one results in a high error and poor efficiency. Palletizing process can resolve these problems.

A full palletizing process includes pallet parameters setting and pallet programming. After you set the pallet parameters on DobotSCStudio, the generated configuration file will be imported to the robot system automatically, then you can write a pallet program by calling pallet API based on site requirements.

3.2.2 Setting Pallet

Pallet parameter settings include basic parameter setting and path point setting. Basic parameter setting is to set pallet name, stack number, palletizing direction and stack spacing. Path points are the configured points on the assembly path or dismantling path.

- Transition point (point A): A point the robot must move to when assembling or dismantling stacks, which is fixed or varies with the pallet layer.
- Preparation point (point B): A point calculated by the target point and the set offset.
- Target point (point C): The first stack point.

Figure 3.24 and Figure 3.25 show the assembly path and dismantling path.

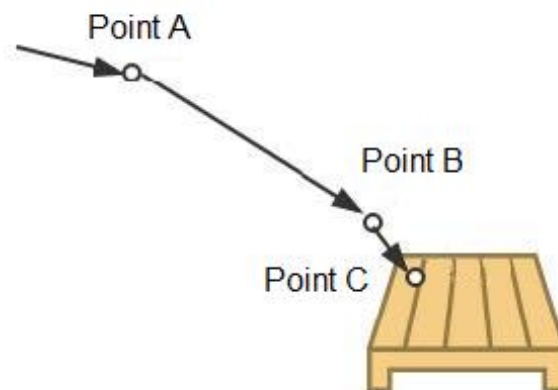


Figure 3.24 Assembly path

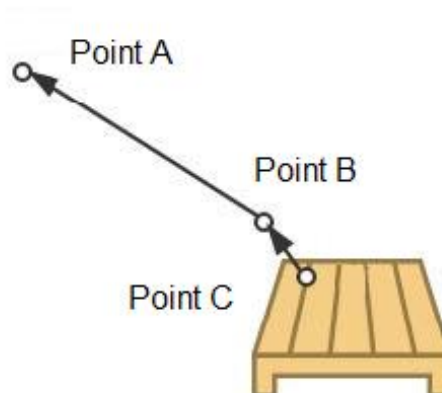


Figure 3.25 Dismantling path

Stack indicates parts or products to be carried. Pallet indicates an object which places the stacks. Assembling stack indicates that the robot places stacks to the pallet as the configured pallet type. Dismantling stack indicates that the robot takes out stacks from the pallet as the configured pallet type. Pallet type indicates the layout of all stacks placed on the pallet. In our robot system, only the matrix pallet is supported, on which the stacks are placed in regular intervals, as shown in Figure 3.26.

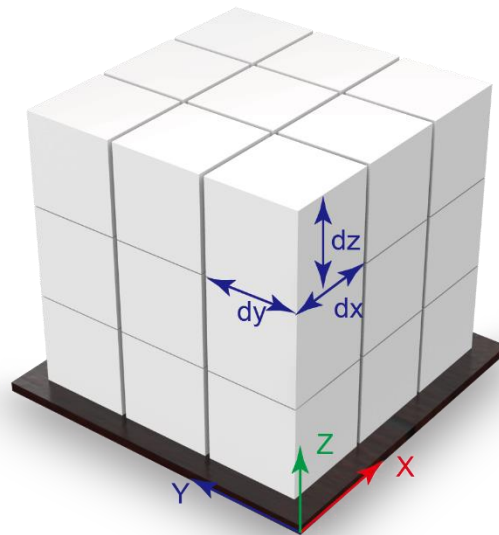


Figure 3.26 Matrix pallet

In this topic, we describe how to set pallet parameters.

Prerequisites

- The robot has been powered on.
- The suction cup or gripper kit has been mounted on the robot
- (Optional) The User coordinate system has been set on the pallet. When teaching positions, you can select the set User coordinate system based on site requirements.

Procedure

Step 1 Click **Process>MatrixPallet**.

The pallet page is displayed, as shown in Figure 3.27.

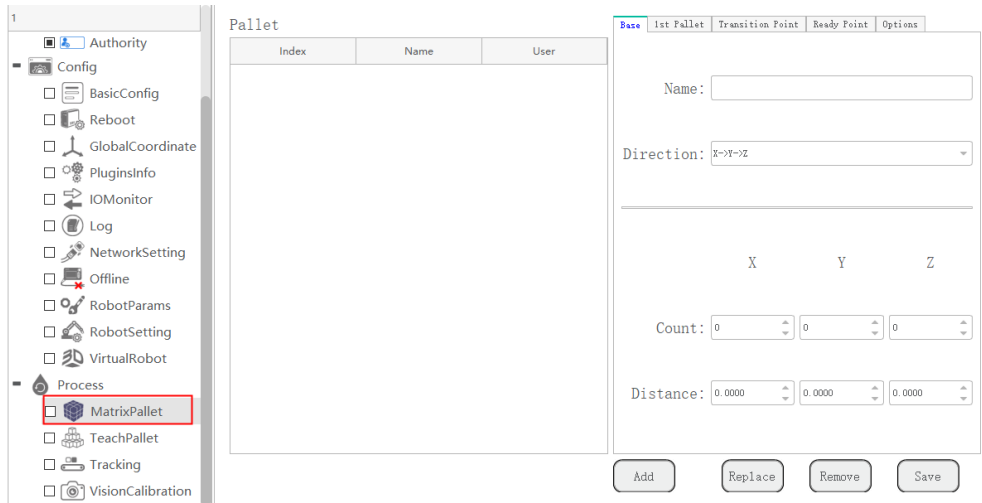


Figure 3.27 Pallet page

Step 2 Set the basic pallet parameters on the **Base** tab.

Table 3.5 shows the basic pallet parameter description.

Table 3.5 Basic pallet parameter description

Parameter	Description
Name	Pallet name
Direction	Palletizing direction Value: X->Y->Z or Y->X->Z In this topic, we select X->Y->Z
Count	Number of stacks in X, Y, Z direction respectively
Distance	Stack interval in X, Y, Z direction respectively

Step 3 Jog the robot to the first stack position and click **Get Pose** on the **1st Pallet** tab, as shown in Figure 3.28.

UserCoord is the User coordinate system index, which needs to be consistent with the User coordinate system selected during teaching.

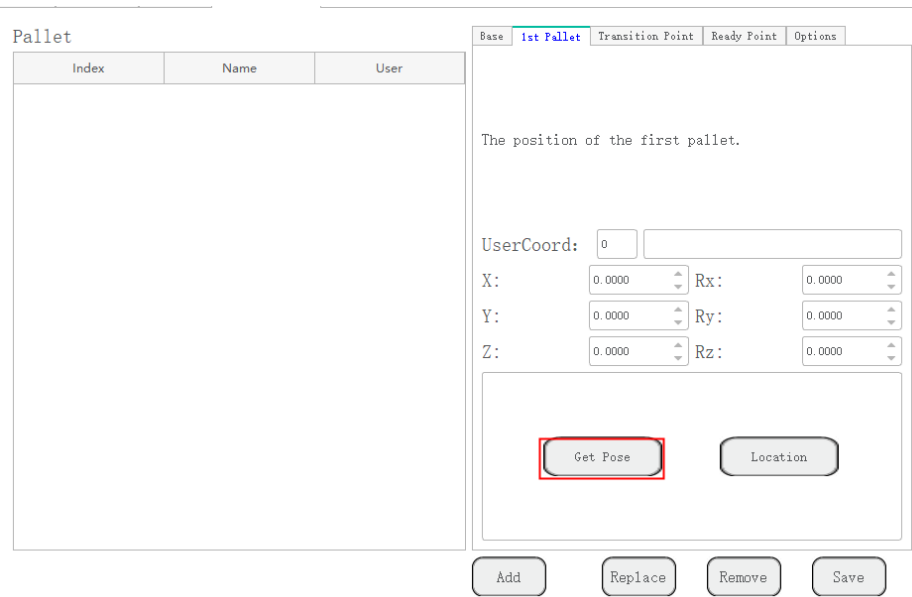


Figure 3.28 Teach the first stack position

Step 4 Jog the robot to the transition point and click **Get Pose** on the **Transition Point** tab, as shown in Figure 3.29.

UserCoord is the User coordinate system index, which needs to be consistent with the User coordinate system selected during teaching.

If **Variation with layer height** is selected, the transition point is varied with the pallet layer. If not, it is the fixed point.

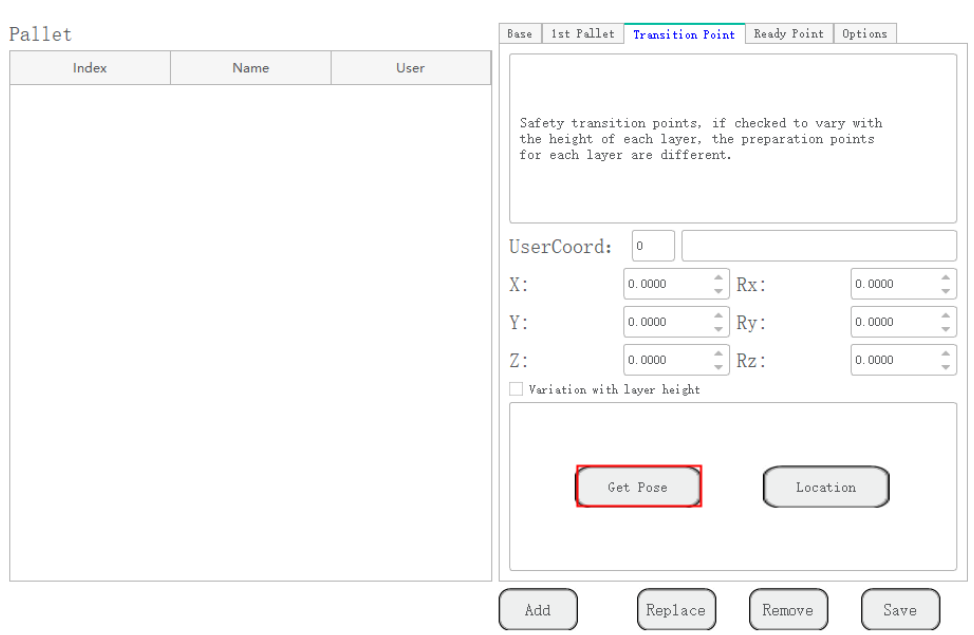


Figure 3.29 Teach the transition point

Step 5 Jog the robot to the position where is above the first stack, and click **Get Pose** on the **Ready Point** tab.

UserCoord is the User coordinate system index, which needs to be consistent with the User coordinate system selected during teaching.

Step 6 Click **Add** to generate the configuration file and import to the robot system automatically.

3.2.3 Example

After setting the pallet parameters, you can call pallet API for programming. This topic takes stack assembly as an example to describe.

Program 3.2 Stack assembly demo

```
local MPpick = MatrixPallet(0, "IsUnstack=true Userframe=8")           // Define the pallet instance
Reset(MPpick)                                                         // Initial the pallet instance
while true do
    MoveIn(MPpick,"velAB=90 velBC=50")                                // Start to assemble
    MoveOut(MPpick)
    result=IsDone(MPpick)
    if (result == true)                                              // Check whether stack assembly is
complete
    then
        print("EXIT ...")
        break
    end
end
end
Release(MPpick)                                                       // Release pallet instance
```

4. Typical Applications

4.1 Modbus Application

A robot can communicate with external equipment by the Modbus protocol. Here, External equipment such as a PLC is set as the Modbus master, and the robot system is set as the slave.

This topic takes a PLC as an example to control the robot by reading and writing related registers. The typical connection is shown in Figure 4.1.

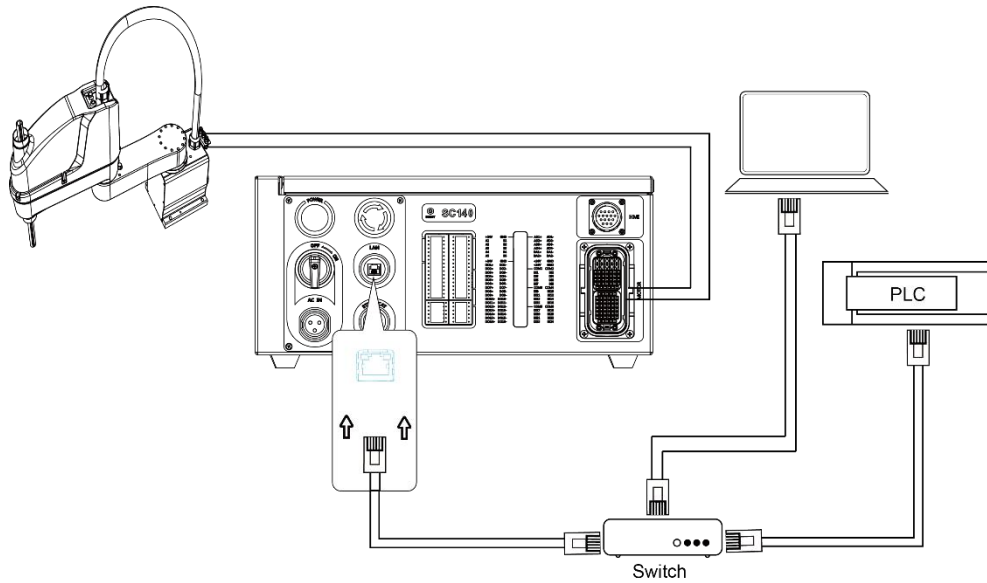


Figure 4.1 Typical connection

The IP address of the robot system must be in the same network segment of the external equipment without conflict. You can modify the IP address on the **Config> NetworkSetting** page; the default port is 502 and cannot be modified.

Figure 4.2 shows the program process and Program 4.1 shows the corresponding demo.

NOTE

- The registers mentioned in Figure 4.2 are the robot system's registers. The corresponding registers of PLC are shown in *2.15.1 Modbus Register Description*.
- This topic only describes the program demo of the robot system. The details on the program of PLC are not described in this topic.

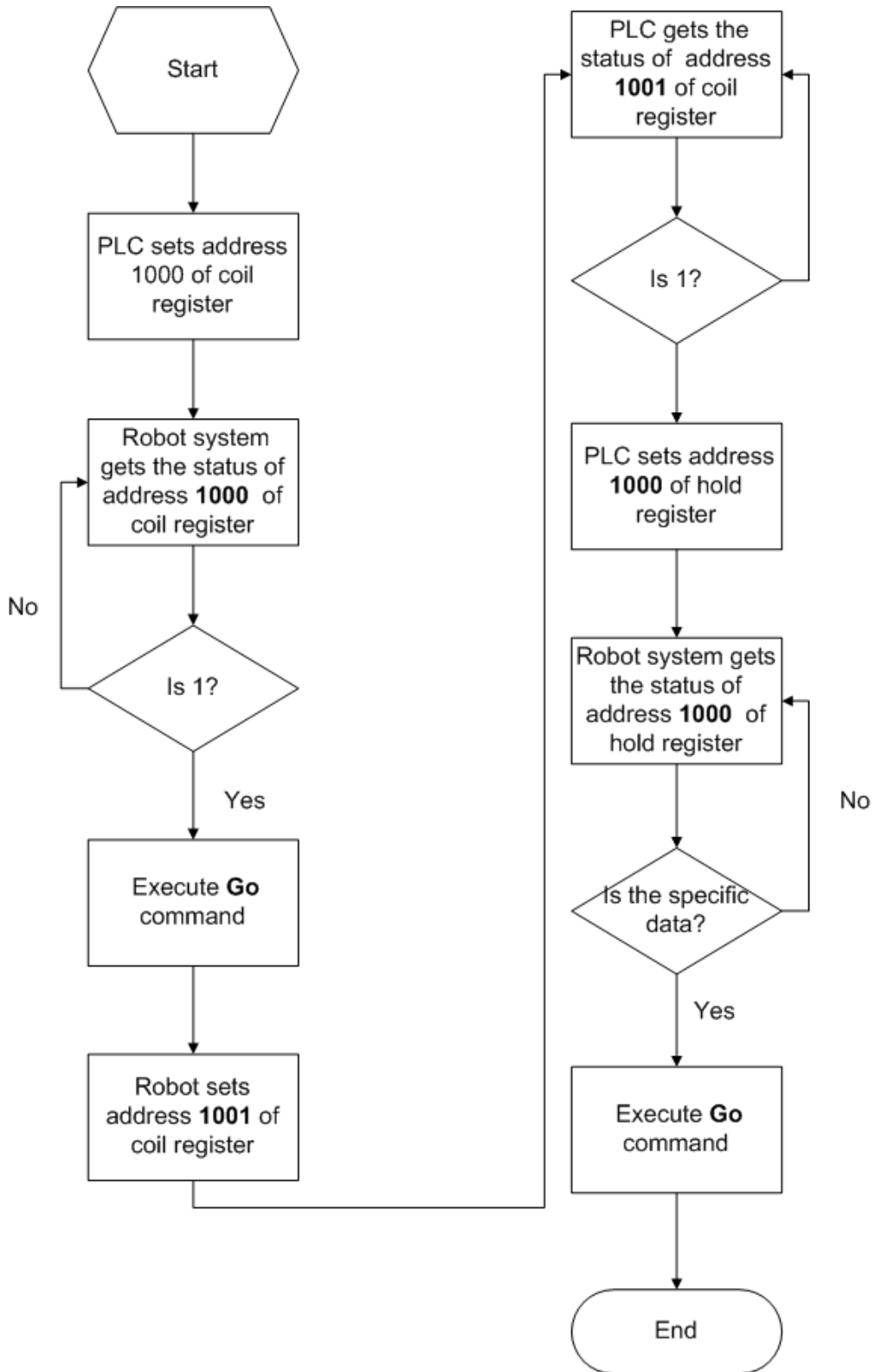


Figure 4.2 Program process

Program 4.1 Slave program demo

local data = { 1 }

```

local data2 = {1}
local coil = {}
local RST = {0}
while true
do
    coil = GetCoils(1000,1)
    if coil[1] == 1 then
        Go(P1,"SYNC=1")
        Sleep(1000)
        SetCoils(1001, 1, data)
        while true do
            Sleep(1000)
            data2 = GetHoldRegs(1000,1)
            if data2[1] == 666 then
                Go(P2,"SYNC=1")
                SetHoldRegs(1001,#data,data2,"U16")
                Sleep(1000)
                Go(P3,"SYNC=1")
                SetCoils(1000, 1, RST)
                SetCoils(1001, 1, RST)
                SetHoldRegs(1000,1,RST,"U16")
                coil = {0}
                data2 = {0}
                break
            end
        end
    end
end
end
end
end

```

4.2 I/O Application

4.2.1 Grabbing Bottle Application

This topic takes a six-axis robot as an example to describe how to grab bottles. The full process includes bottle transportation, bottle grabbing, box covering, and box transportation. The bottle and box transportation is completed by the conveyor and PLC. Bottle grabbing is completed by the robot. In this topic, we only describe how to grab, as shown in Figure 4.3.

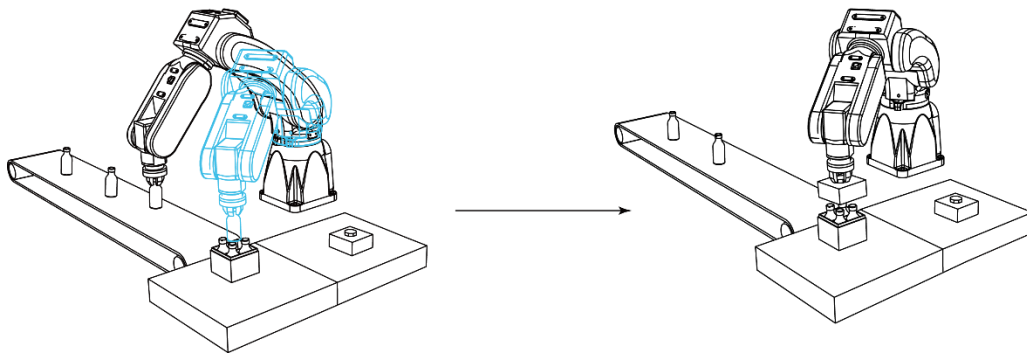


Figure 4.3 Application scene

Table 4.1 lists the positions where the robot will move to.

Table 4.1 Position description

Position	Description
P1	Safe position
P2	Bottle grabbing transition position
P3	Position closed to the bottle grabbing position
P4	Bottle grabbing position
P5, P7, P9, P11	Position above the placing position
P6, P8, P10, P12	Placing position
P13	Lid grabbing transition position
P14	Lid grabbing position
P15	Position above the lid grabbing position
P16	Position above the lid covering position
P17	Lid covering position

Each time the robot performs a task, it sends a signal to the PLC to identify the task that needs to be completed so that the PLC can perform the related work. Table 4.2 lists the signal descriptions.

Table 4.2 User-defined digital signal description

Digital signal	Description
Input	
DI2	Allow to grab a bottle

Digital signal	Description
DI3	Allow to grab a lid
Output	
DO7	Return to the safe position
DO8	Complete bottle grabbing
DO9	Cover a lid
DO17	Grab a bottle
DO18, DO19	Grab a lid

Figure 4.4 shows the detailed process.

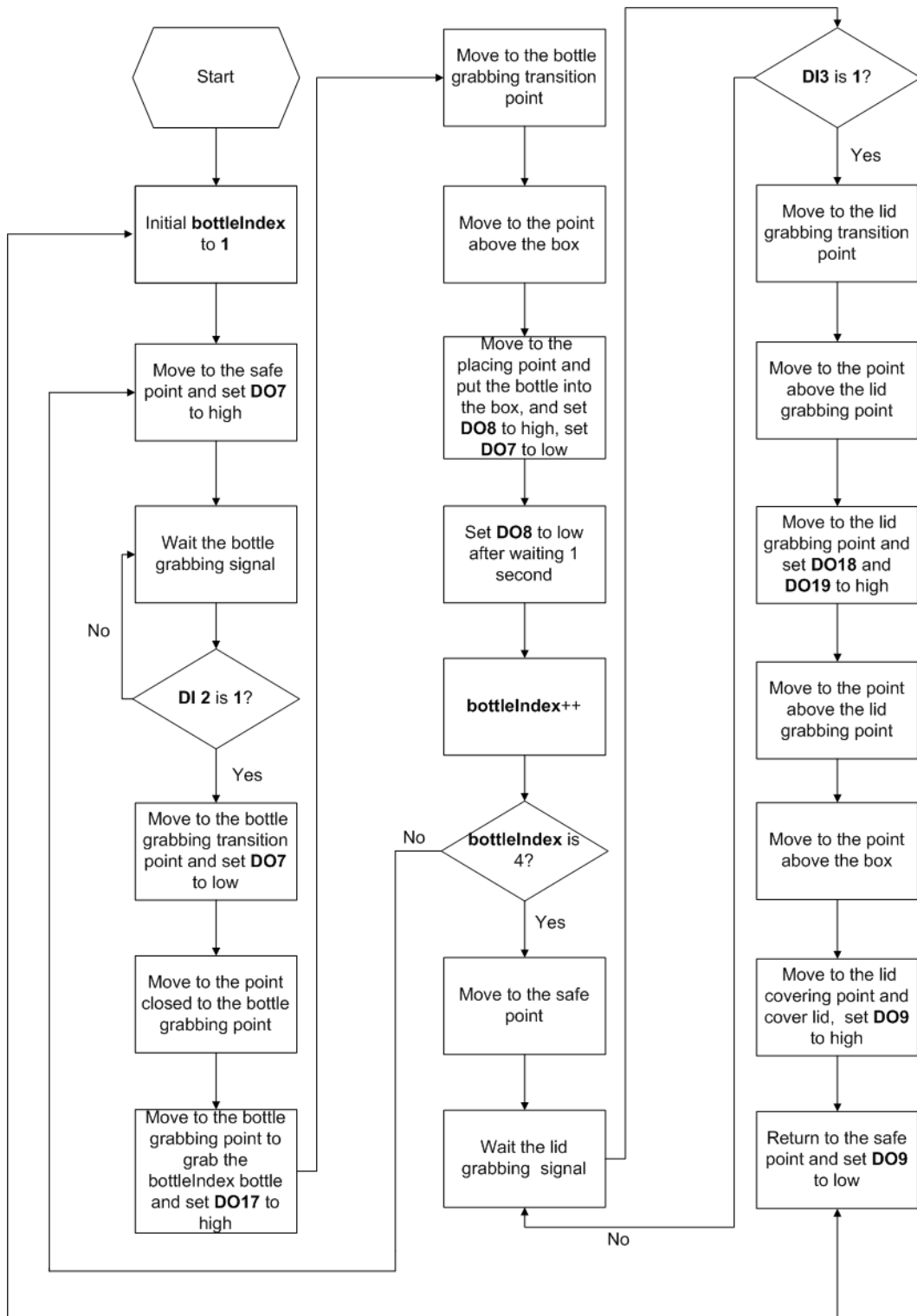


Figure 4.4 Grabbing bottle process

- Program 4.2 Grabbing bottle demo

local bottleNum = 4

local readyPos=P1 --Safe position

```

local bottleLoadingPos=P4 --Bottle grabbing position
local bottleUnloadingPosup={P5, P7, P9, P11}
local bottleUnloadingPos={P6, P8, P10, P12}
local boxLoadingPos = P14 -- Lid grabbing position
local boxUnloadingPos = P17 -- Lid covering position
while true do
    for bottleIndex = 1, bottleNum do
        Go(readyPos,"Speed=100 Accel=100")
        DOExecute(7,1)
        Wait(1000)
        while true do --Grab bottle signal
            Sleep(500)
            input = DI(2,1)
            if input == 1 then
                break
            end
        end
        end
        DOExecute(7,0)
        Wait(500)
        Go(P2,"Speed=100 Accel=100 SYNC=1") -- Bottle grabbing transition position
        Move(P3,"Speed=100 Accel=100 SYNC=1") --Above the bottle
        Move(bottleLoadingPos," Speed=100 Accel=100 SYNC=1") --Grab bottle position
        DOExecute(17,1) --Grabbing signal
        Wait(500)
        Move(P3,"Speed=100 Accel=100 SYNC=1") --Above the bottle
        Go(P1,"Speed=100 Accel=100 SYNC=1") --Transition position
        Go(bottleUnloadingPosup[bottleIndex],"Speed=100 Accel=100 SYNC=1") --Above the bottle
        Move(bottleUnloadingPos[bottleIndex],"Speed=100 Accel=100 SYNC=1") --Bottle location
        DOExecute(17,0) --Grabbing signal
        Wait(500)
        Move(bottleUnloadingPosup[bottleIndex],"Speed=100 Accel=100 SYNC=1")
        DOExecute(8,1) --Grab the bottle to complete the signal
        Wait(1000)
        DOExecute(8,0)
    end
    end
    Go(readyPos,"Speed=100 Accel=100 SYNC=1") --Safe position
    while true do -- Grab the lid signal

```

```
Sleep(1000)
input = DI(3,1)
if input == 1 then
    break
end
end
end
Go(P13,"Speed=100 Accel=100 SYNC=1") --Transition position
Move(P15,"Speed=100 Accel=100 SYNC=1") --Grab the lid transition position
Move(boxLoadingPos," Speed=100 Accel=100 SYNC=1") --Grab the lid
DOExecute(18,1) --Grabbing signal
DOExecute(19,1) --Grabbing signal
Wait(500)
Move(P15,"Speed=100 Accel=100 SYNC=1") --Grab the lid transition position
Move(P16,"Speed=100 Accel=100 SYNC=1") --Cover the box and cover it transition position
Move(boxUnloadingPos,"Speed=10 Accel=10 SYNC=1") --Cover the box and cover it
DOExecute(18,0)
DOExecute(19,0)
Wait(500)
DOExecute(9,1)
Wait(500)
Move(P16,"Speed=100 Accel=100 SYNC=1") --Cover the box and cover it transition position
Go(readyPos,"Speed=100 Accel=100 SYNC=1") --Safe position
DOExecute(9,0)
Wait(500)
end
```