Dobot CR Series Robot

APP User Guide

Issue: V3.7

Date: 2020-12-04

Shenzhen Yuejiang Technology Co., Ltd

Precautions

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

Website: www.dobot.cc

Preface

Purpose

This Document describes how to use Dobot CR series robots with APP, making it easy for users to fully understand and use it.

Intended Audience

This document is intended for:

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

Change History

Date	Change Description
2020/12/04	The second release Add palletizing process
2020/11/17	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 of Software

The robot supports Android/iOS tablet operation and PC operation. This manual uses Android tablet as an example.

The interface is shown in Figure 1.1, and its detailed description is shown in Table 1.2.

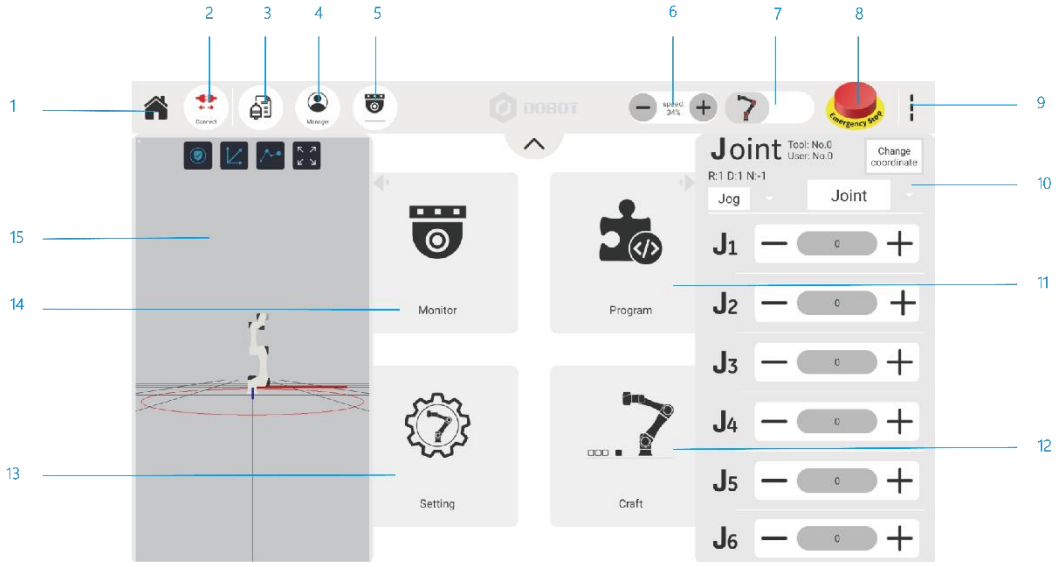

Figure 1.1 Homepage

Table 1.1 Cable color description

NO.	Description
1	Click this button to go back to the previous page
2	Connection button When device and robot arm are connected to network, click the button to connect the device to robot arm
3	Alarm log You can click it to check alarm log

NO.	Description
4	Manager APP login personnel are divided into observer, operator, programmer and manager, different personnel can operate different functions, including the most authority of the management, can operate all functions • Observer: 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
5	Monitoring module shortcuts
6	Set speed ratio
7	Robotic arm enable button The icon is green when robot motor is in the enabled status Tshe icon is red when robot motor is in the disabled status
8	Emergency stop switch Emergency stop switch can be pressed when robot arm is in short of time during operation, to control servo drive power off and robot arm stop urgently but constant power
9	System settings Click this button to expand the page to view help documentation, lock screen, switch skins, etc.
10	Jogging robotic arm You can jogging the robot arm. Click the button Joint to switch the joint coordinate system or the Cartersin coordinate system
11	Programming module Programming modules are mainly used for editing and running programs
12	Process Support drag teaching, conveyor tracking
13	Setting Set the related parameters of robot arm, including motion parameters, coordinate system settings, calibration, etc.

NO.	Description
14	Monitoring module You can view the status of robot arm, set the output of I/O , set the end parameters of robot arm and other functions
15	The 3D illustration of a robotic arm

NOTICE

The software supports the screen lock function. If the software has not been operated for a long time, the screen is automatically locked. The unlock password is 000000 by default.

1.1 Fast Connection

Prerequisites

- The controller has been connected to the WiFi module.
- The APP supports WiFi function.

Procedure

Step 1 Search Dobot controller WiFi name and connect it. The WiFi name is prefixed with **Dobot_WIFI_XXX**. The default WiFi password is **1234567890**.

You can modify the WiFi password on t**he Setting > Software Set > WiFi Set** page with manager authority.

Step 2 Click **Connect** on the left pane of the APP.

1.2 Setting

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

1.2.1 Set Jog

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

Set the maximum velocity and acceleration in Joint and Cartesian coordinate system when jogging a robot. As shown in Figure 1.2.

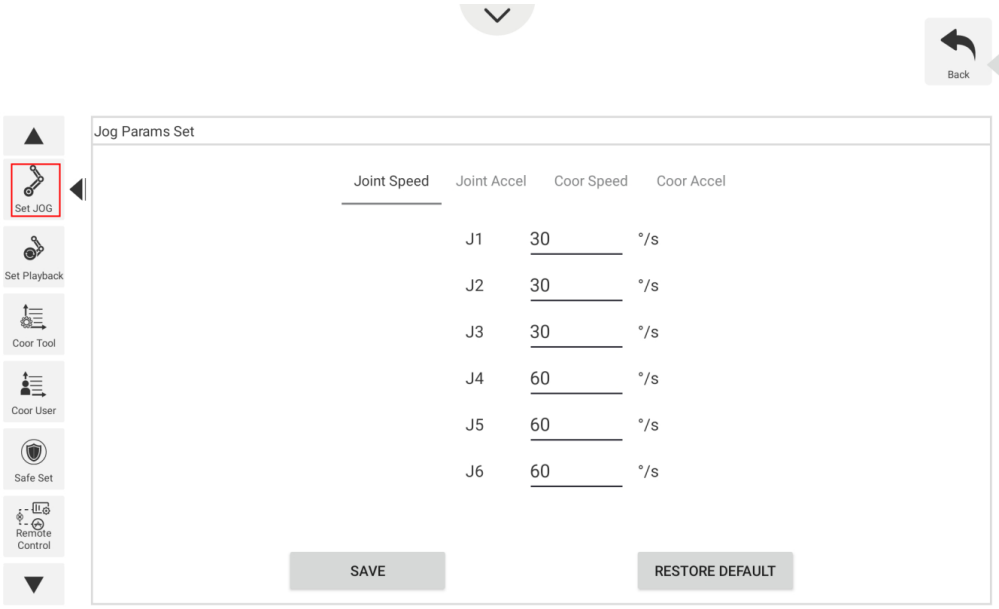

Figure 1.2 Jogging parameters in the Joint coordinate system

1.2.2 Set Playback

Set the maximum velocity, acceleration, and jerk in the Joint and Cartesian coordinate system when running robot programs, as shown in Figure 1.3.

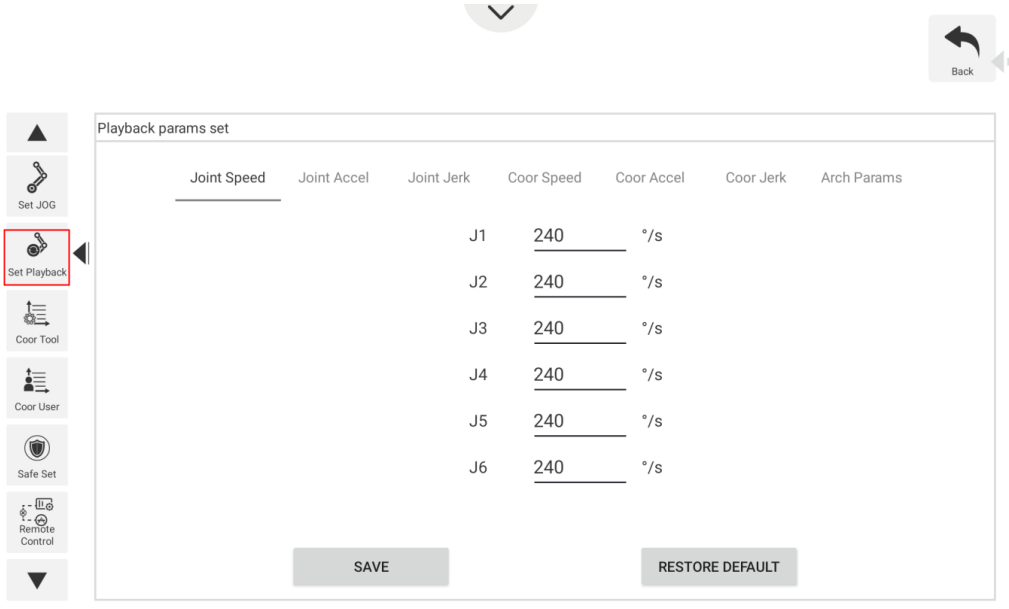

Figure 1.3 Playback parameters

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

NOTE

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.*

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 1.4. Please refer to Table 2.24 for use

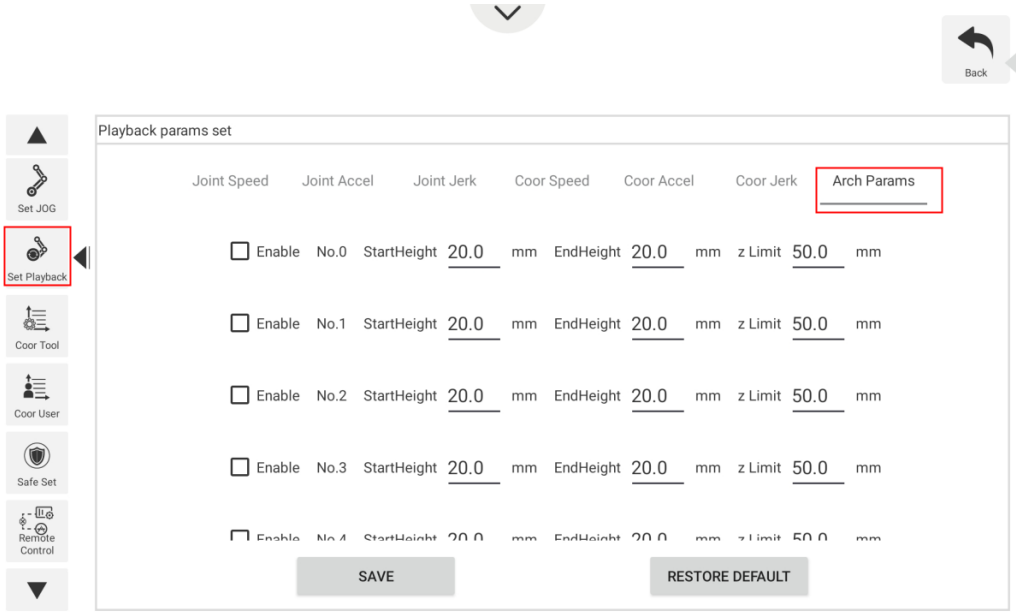

Figure 1.4 Jump parameters

1.2.3 Coor User

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.

- Point: move TCP to any point **A** to create origin, and create user coordinate system according to the default tool coordinate system As shown in Figure 1.5.

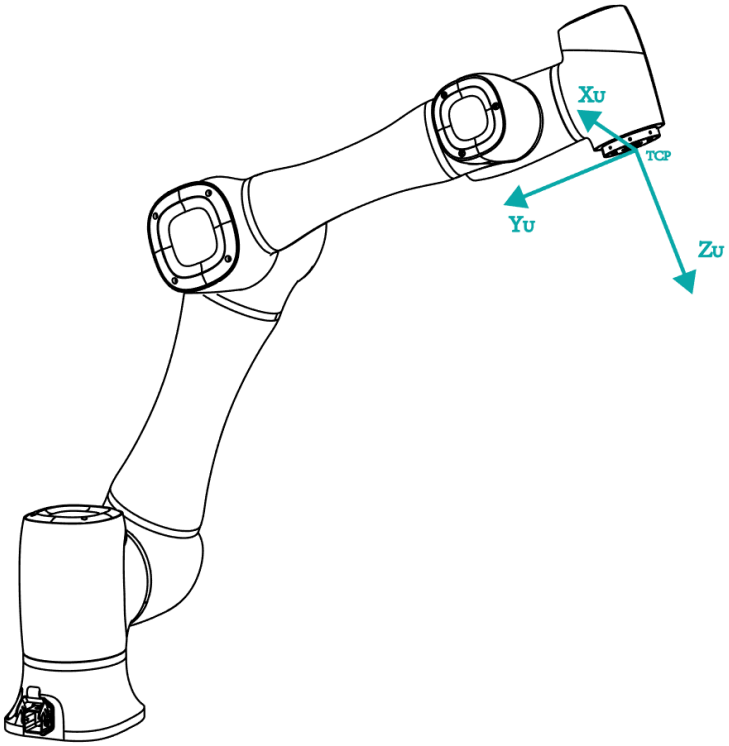

Figure 1.5 Point

- Line: Confirm a straight line by any two points **A** and **B**. The direction from A to B is defined as the positive direction of Y-axis, The Z-axis of Tool coordinate system of which point A is the origin is projected into the vertical plane that confirmed by points A and point B, we can define it as the positive direction of Z-axis. and then the positive direction X axis can be defined based on the right-hand rule. As shown Figure 1.6.

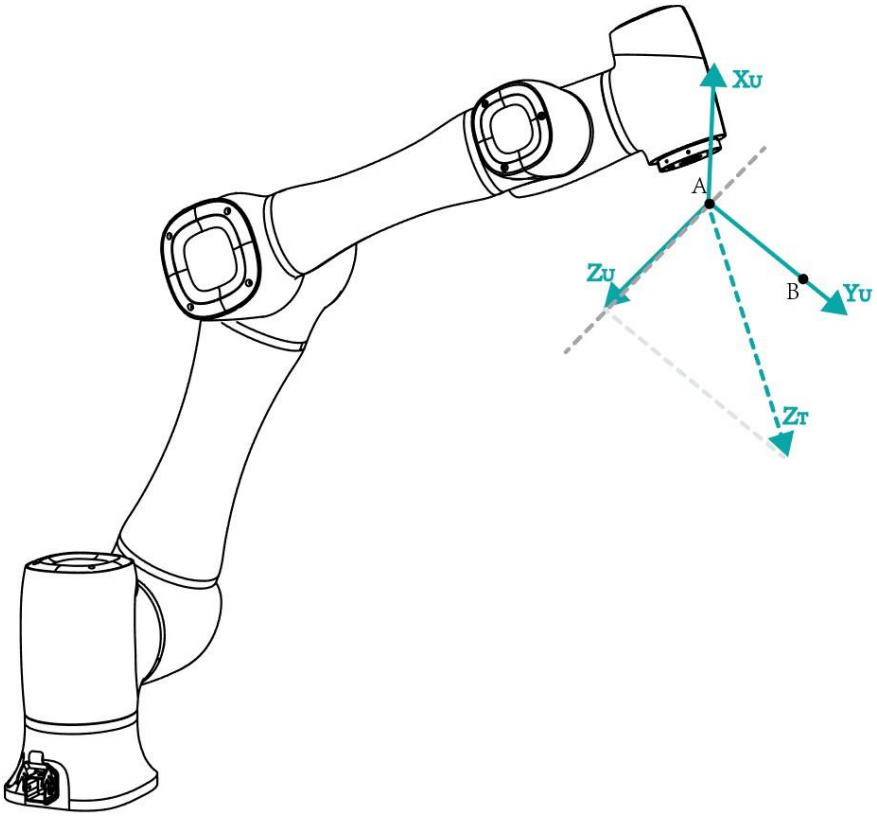

Figure 1.6 Line

- Area: User coordinate system is created by three-point calibration method. Move the robot to three points **A(x1, y1, z1)**, **B(x2, y2, z2)**, and C(**x3**, **y3**, **z3**). 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 1.7.

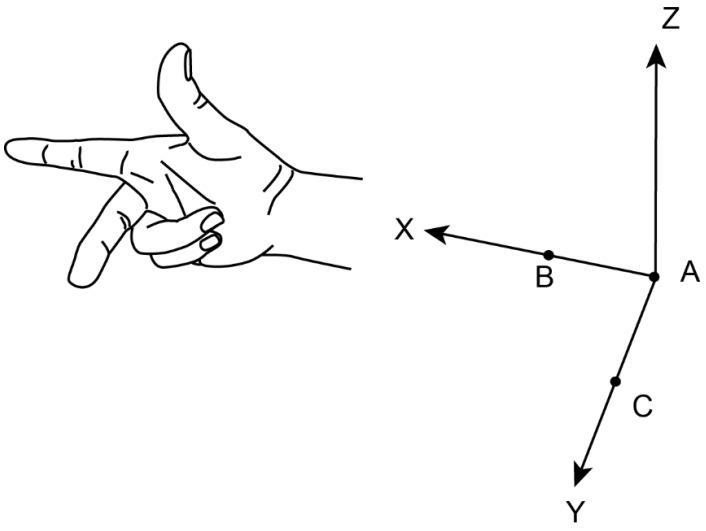

Figure 1.7 Area

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

Prerequisites

- The robot has been powered on.
- The robot is enabled.
- The robot is in the Cartesian coordinate system.
- The user's authority is programmer authority or manager authority.

Procedures

Step 1 Click **Coor User** on the **Setting** page. As shown Figure 1.8.

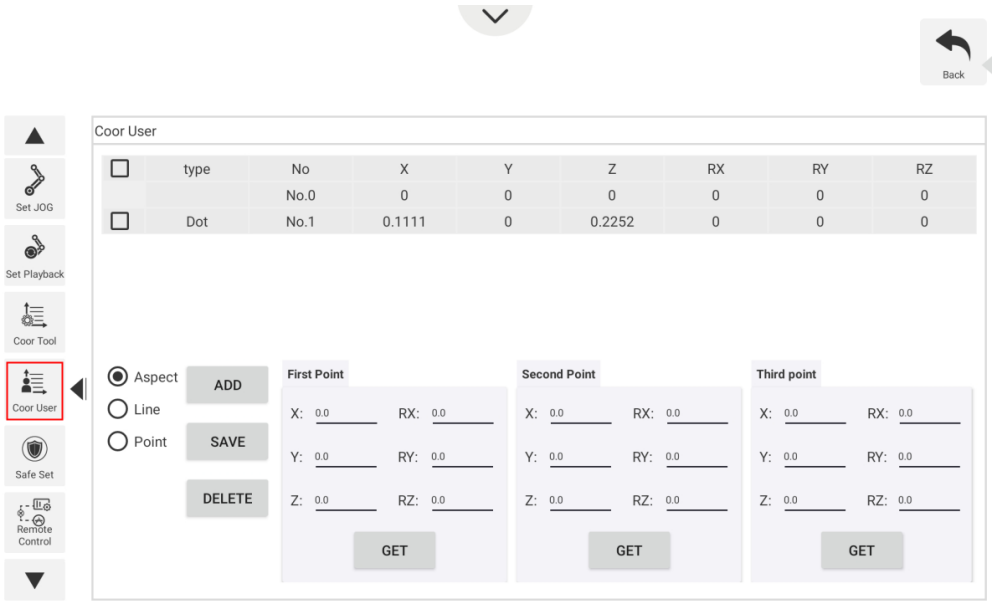

Figure 1.8 User coordinate system

Step 2 Click **Aspect** and jog robot to the first point and click **GET** on the **First Point** section to get the first point.

Step 3 Jog robot to the second point and click **GET** on the **Second Point** to get the second point.

Step 4 Jog robot to the third point and click **GET** on the **Third Point** to get the third point.

Step 5 Click **ADD** and **SAVE** to create User 1coordinate system.

Step 6 Click **Change coordinate** on the APP and select **UserCoordinate** to **NO 1**. If the icon **NO.0** turns into **NO.1**, you can use the User 1 coordinate system for teaching and programming.

1.2.4 Coor Tool

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.

Tool coordinate system 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 1.9.

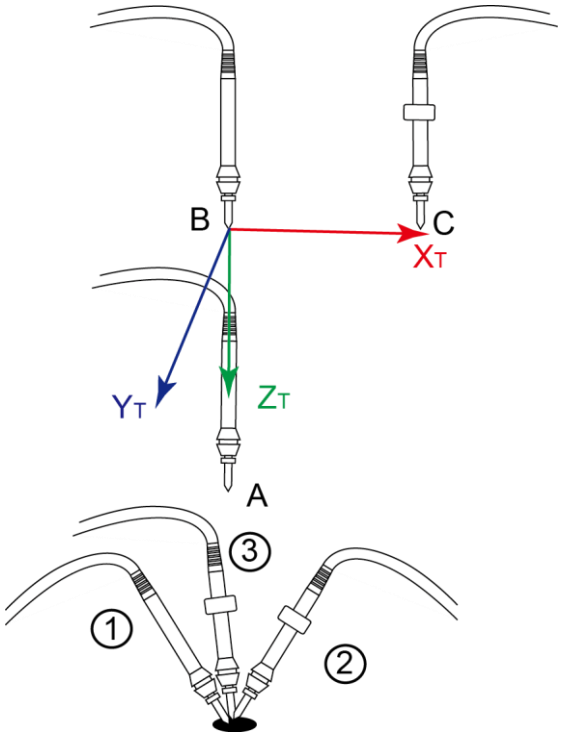

Figure 1.9 Three points calibration method (TCP+ZX)

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

Prerequisites

- The robot has been powered on.
- The robot is enabled.
- The robot is in the Cartesian coordinate system.
- The user's authority is programmer authority or manager authority.

Procedure

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

topic.

Step 2 Click **Tool coordinate** on the **Setting** page. As shown Figure 1.10.

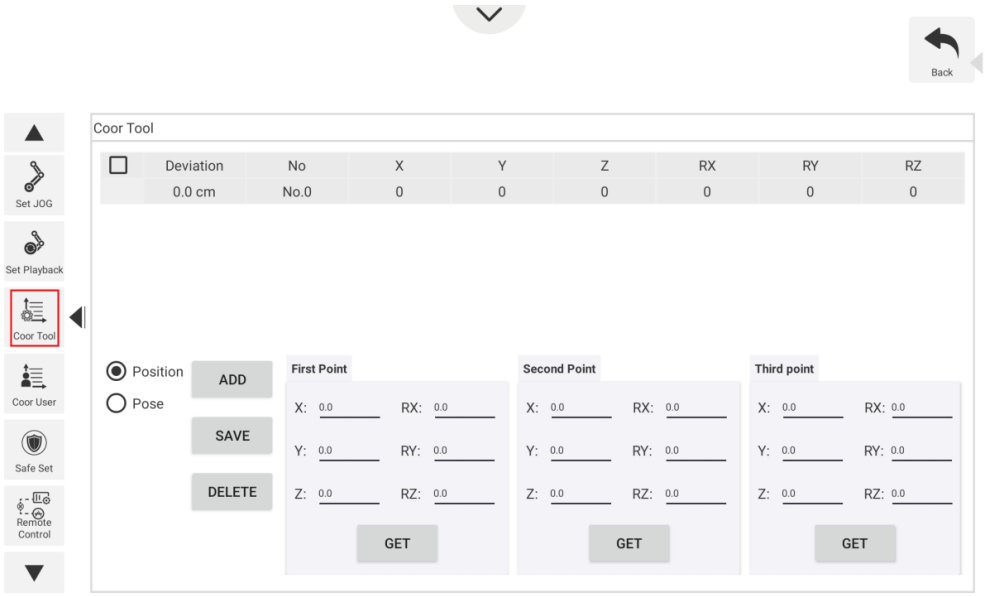

Figure 1.10 Tool Coordinate page

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 Click **Position**, Jog the robot to the reference point in the first direction, then click **GET** on the **First Point** section to get the coordinates of the first point.

Step 4 Jog the robot to the reference point in the second direction, then click **GET** on the **Second Point** section to get the coordinates of the second point.

Step 5 Jog the robot to the reference point in the third direction, then click **GET** on the **Third Point** section to get the coordinates of the third point.

Step 6 Jog the robot to the reference point (point **A)** in the vertical direction, then click **GET** on the **First Point** section to get the fourth point.

Step 7 Jog the Z-axis to a point (point **B**) along the positive direction, then click **GET** on the **Second Point** section to get the fifth point

This step defines the Z-axis.

Step 8 Jog the X-axis to another point (point **C**), then click **GET** on the **Third Point** section to get 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** and **SAVE** to create Tool 1coordinate system.

Step 10 Click **Change coordinate** on the APP and select **ToolCoordinate** to **NO.1**. If the If the icon **NO.0** turns into **NO.1**, you can use the User 1 coordinate system for teaching and programming.

1.2.5 Safe Set

In the safety setting module, you can set safety parameters such as collision detection, power control, joint braking, etc.

This module must be operated with the programmer authority or manager authority.

1.2.5.1 Safe Collision

Collision detection is mainly used for reducing the impact on the robot arm, to avoid damage to the robot arm or external equipment. If the collision detection is activated, the robot arm will stop running automatically when the robot arm hits an obstacle.

You can enable collision detection function on the **Setting > Safe Set > Safe Collision** page and set the collision level. Meanwhile, you can select **Turn on automatic drag after collision**, namely, when the robot arm stops running after hitting an obstacle, you can drag robot to a safe position.

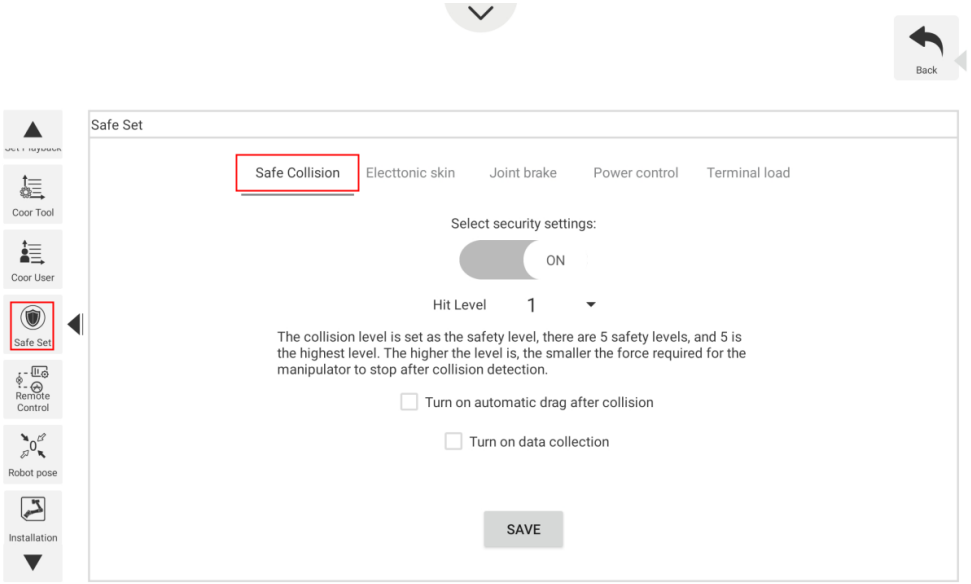

Figure 1.11 Safety Hit

1.2.5.2 Electronic Skin

Electronic skin allows robot to respond in real time when robot meets an obstacle, helping robot avoid obstacle during running.

You can enable electronic skin function on the **Setting > Safe Set > Electronic skin** page and set the robot status when meeting an obstacle. For example, robot can bypass the obstacle or stop running. You can also set the electronic skin parameters on this page

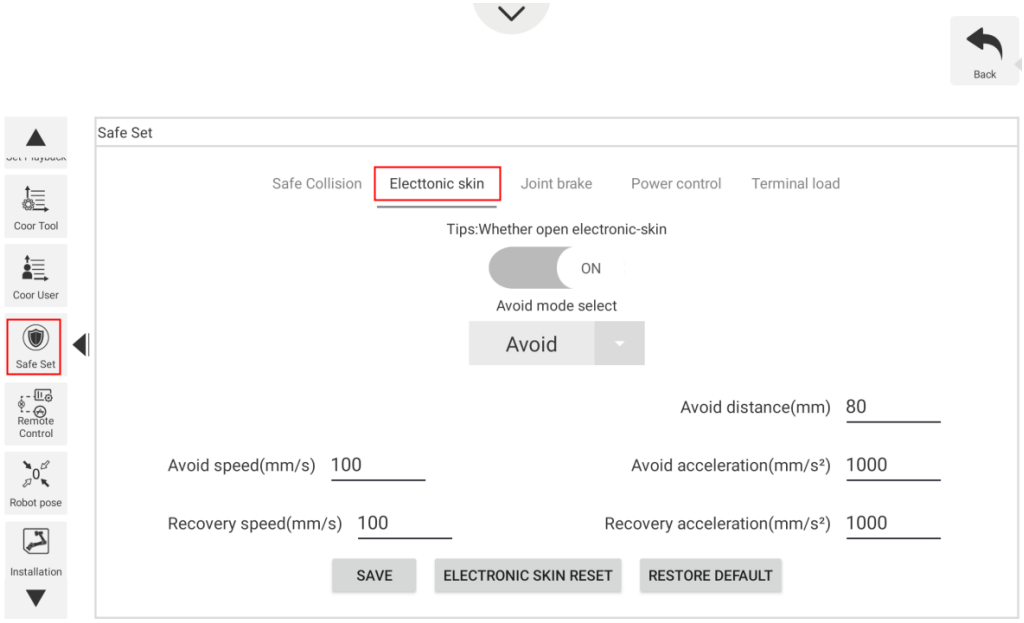

Figure 1.12 Safety Hit

Parameter	Description
Avoid speed	Range: 1~500, recommend value: 100
Avoid distance	Range: 0~200, recommend value: 80
Avoid acceleration	Range: 1~50000, recommend value:1000
Recovery speed	Range: 1~500, Recommend value: 100
Recovery acceleration	Range: 1~50000, Recommend value: 1000

1.2.5.3 Joint Brake

If you want to drag joints by hand, please enable the braking function. Namely, Open each joint brake on the **Setting** > **Safe Set** > **Joint brake** page. When enable this function, please hold the robot arm to avoid damage.

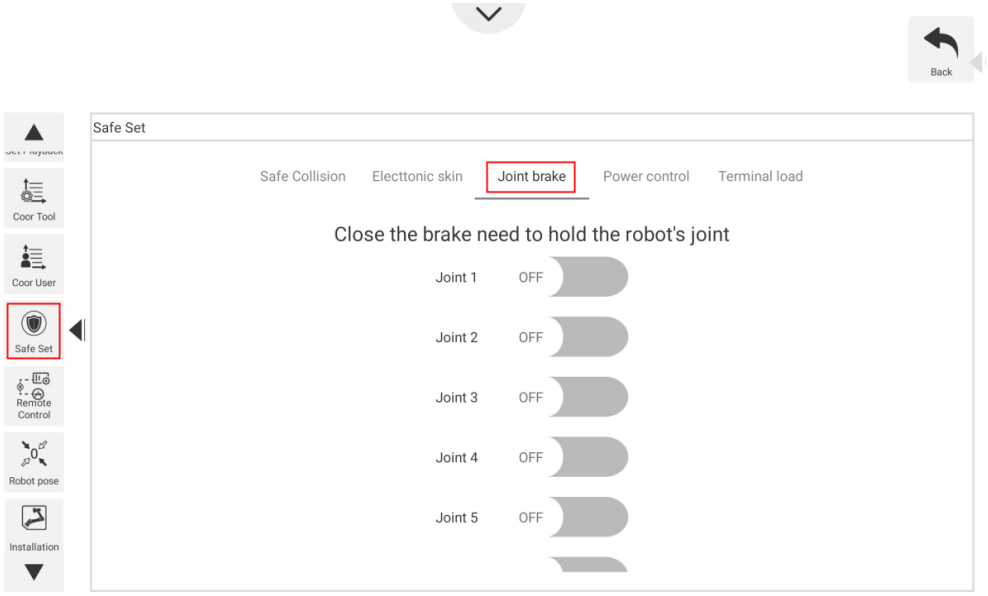

Figure 1.13 Joint brake

1.2.5.4 Power Control

When the emergency stop switch is pressed, the robot will power off. You can click Power On on the **Setting > Safe Set > Power control** page to power on the robot. Also, you can power off the robot on this page.

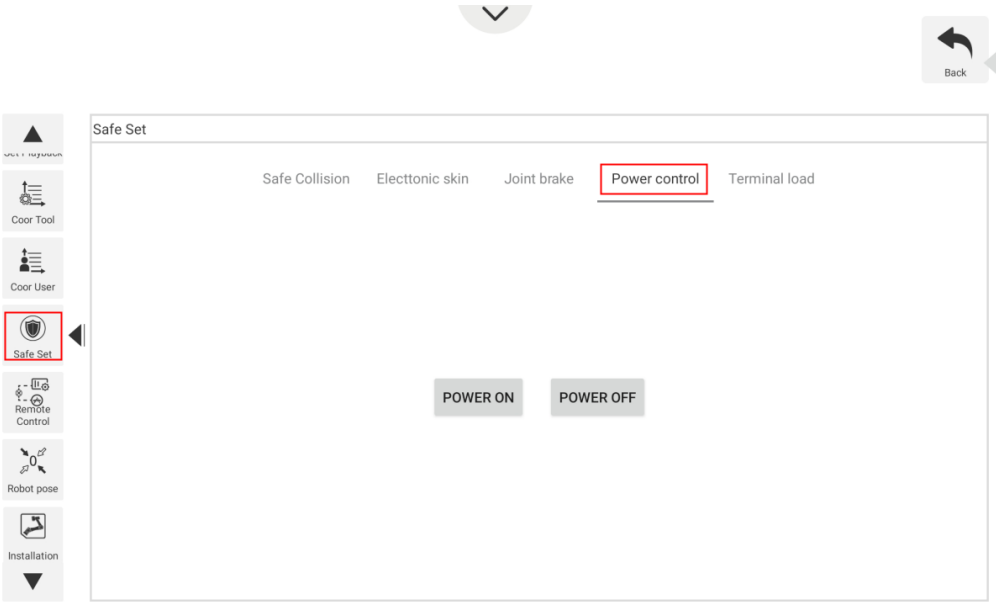

Figure 1.14 Power control

1.2.5.5 Terminal Load

To ensure optimum robot performance, it is important to make sure the load and inertia of the end effector are within the maximum rating for the robot.

The load is weight of the end effector and work piece, which must not exceed the maximum load. You can set the load on the **Setting** > **Safe Set** > **Terminal load** page.

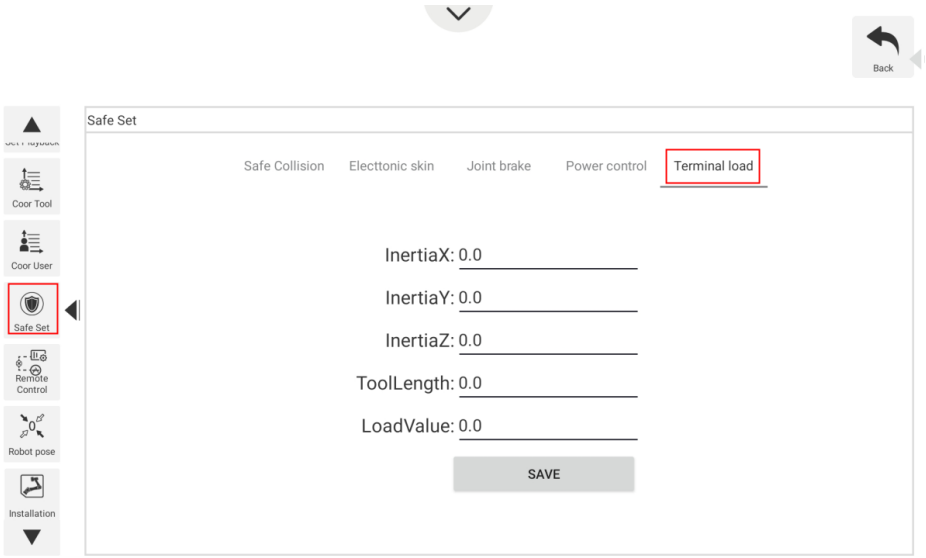

Figure 1.15 Terminal load

1.2.6 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 APP 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.

1.2.6.1 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

I/O interface	Description
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 low to high.

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.
- The user's authority is programmer authority or manager authority.

NOTE

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

Procedure

Step 1 Click **Setting > Remote Control**.

The remote control page is displayed, as shown in Figure 1.16.

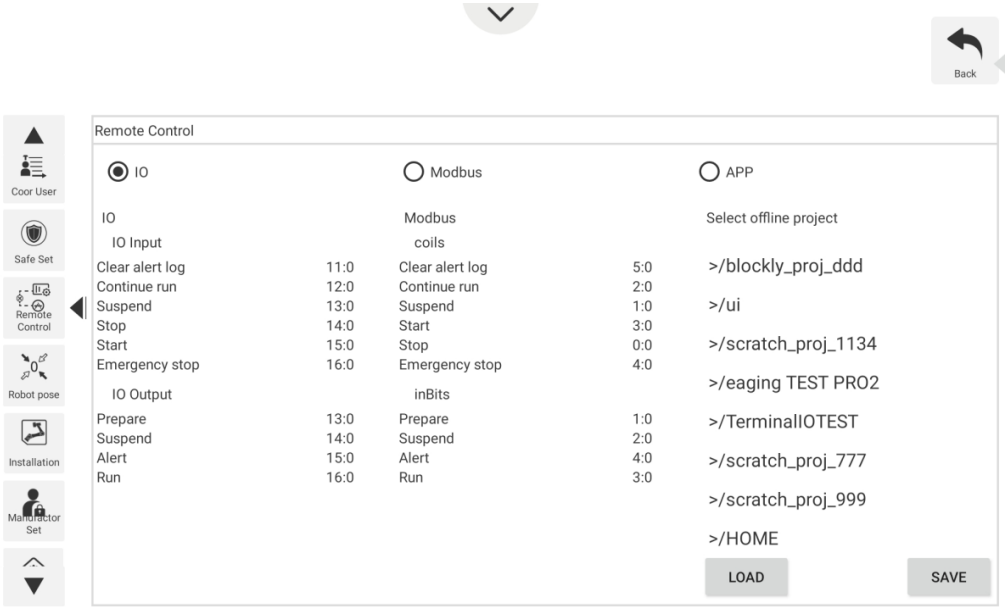

Figure 1.16 Remote control page

Step 2 Select **IO** on the **Remote Control** page and select the offline project on the **Select Offline Project** section. Click **SAVE.**

Right now, only the emergency stop button and the real-time coordinates displaying section are 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.

1.2.6.2 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 *2.16.1Modbus Register Description.*

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

Register address (Take a PLC as an example)	Register address (Robot system)	Description
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. The default IP address is 192.168.5.1.
- The robot has been powered on.
- The user's authority is programmer authority or manager authority.

  NOTE

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

Procedure

Step 1 Click **Setting > Remote Control**.

The remote control page is displayed, as shown in Figure 1.17.

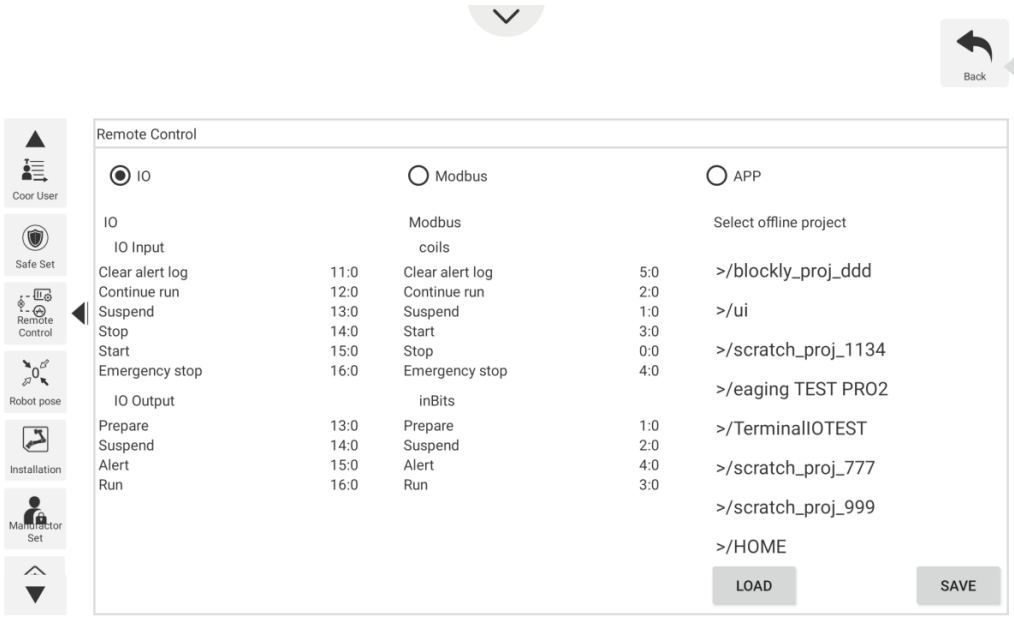

Figure 1.17 Remote control page

Step 2 Select **Modbus** on the **Remote Control** page and select the offline project on the **Select Offline Project** section. Click **SAVE** and **LOAD**.

Right now, only the emergency stop button and the real-time coordinates displaying section are 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.

1.2.7 Robot Pose

APP supports to move robot to common points: package point, homing point, user-defined point. Moving robot to the package point can reduce the robot space, easy to pack and transport. Homing point is homing position. User-defined point is user-defined based on site requirements, which is convenient to move to this position quickly.

This module must be operated with the programmer authority or manager authority.

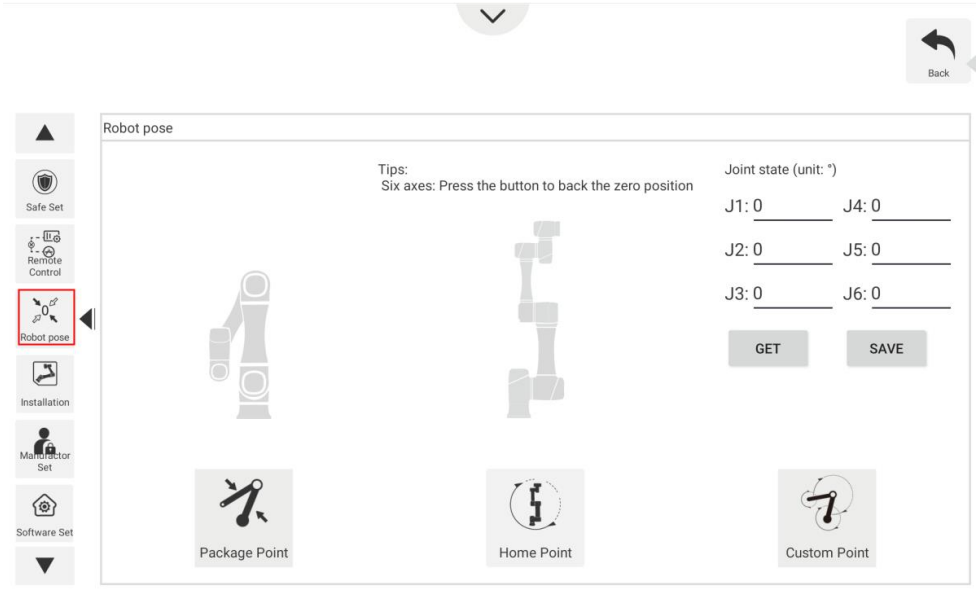

Figure 1.18 Posture setting

- Package Point: Click 

to move robot to the factory point.

- Home Point: Long press 

to move robot the homing point.

- Custom Posture: Click 

to move robot to the user-defined point.

Before moving to the user-defined point, you need to define this point: Jog robot to this point, click **Get** and **Save**.

1.2.8 Installation

The default is that the robot is mounted on a flat table or floor, in which case no change is needed on this page. However, if the robot is ceiling mounted, wall mounted or mounted at an angle, you need to set the rotation angle and slop angle in the disabled status. This module must be operated with the manager authority.

Also, you can click **CALIBRATE** on the **Installation** page after installing the robot and enabling the robot motor, and operate the robot based on the tips(Jog robot in the joint coordinate system to make the end flange perpendicular up to the ground) shown on the **Installation** page, then click **Calibrate installation angle** to obtain the rotation angle and slope angle.

Slop angle is the angle that robot rotates counterclockwise around X-axis at the origin point.

Rotation angle is the angle that robot rotates counterclockwise around Z-axis at the origin point.

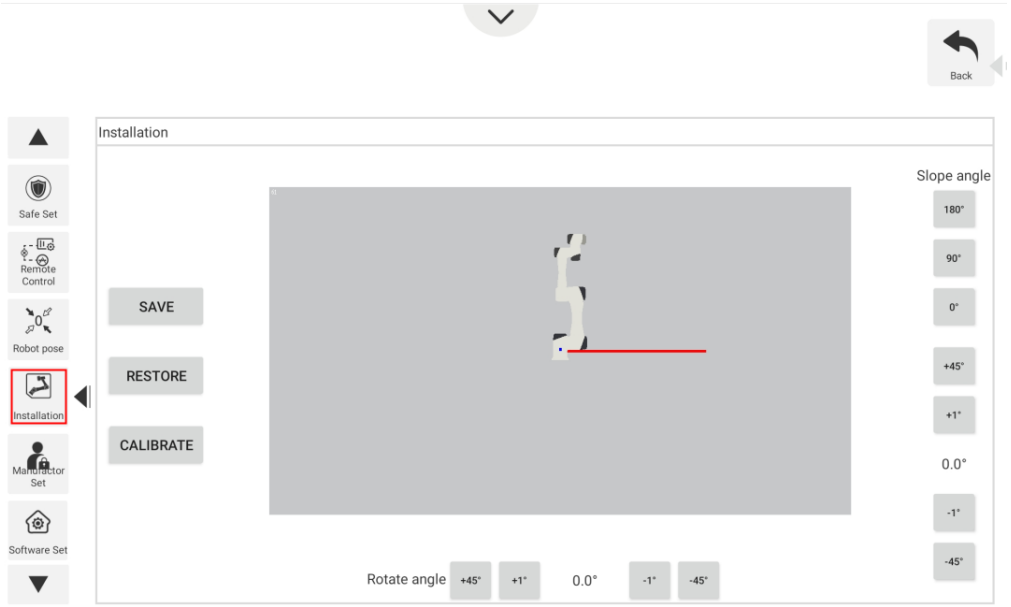

Figure 1.19 Installation setting

1.2.9 Manufactory Set

Generally, this module is used by Dobot support engineer or factory member with manager authority. The details will not be described in this topic.

1.2.10 Software Set

This module is operated with manager authority.

1.2.10.1 Lock Set

You can set APP lock screen time or modify the locking password based on site requirements. If the software is not operated within this time period, the system will automatically lock the screen. The unlock password defaults to: 000000.

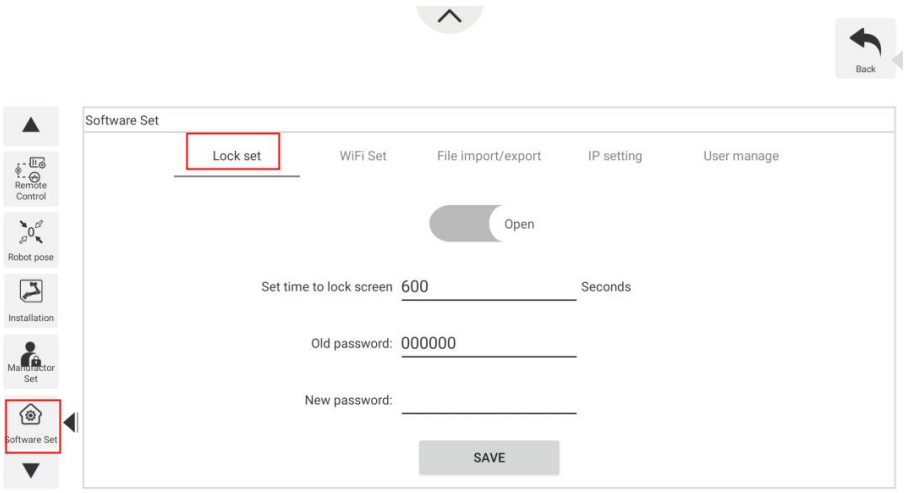

Figure 1.20 Setting lock time

1.2.10.2 File Import/Export

This function is used to import or export file from controller which can improve reusability.

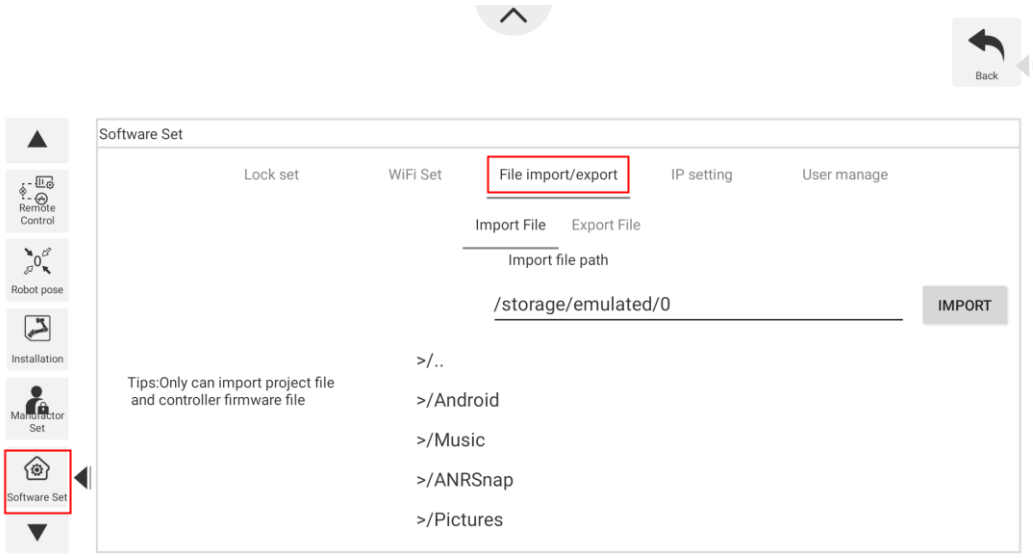

Figure 1.21 Import/export file

1.2.10.3 WiFi Set

The robot system can be communicated with external equipment by the WiFi module. You can modify the WiFi name and password on the **Setting > Softeare Set > WiFi Set** page and then restart the controller to make effective. The default password is **1234567890**.

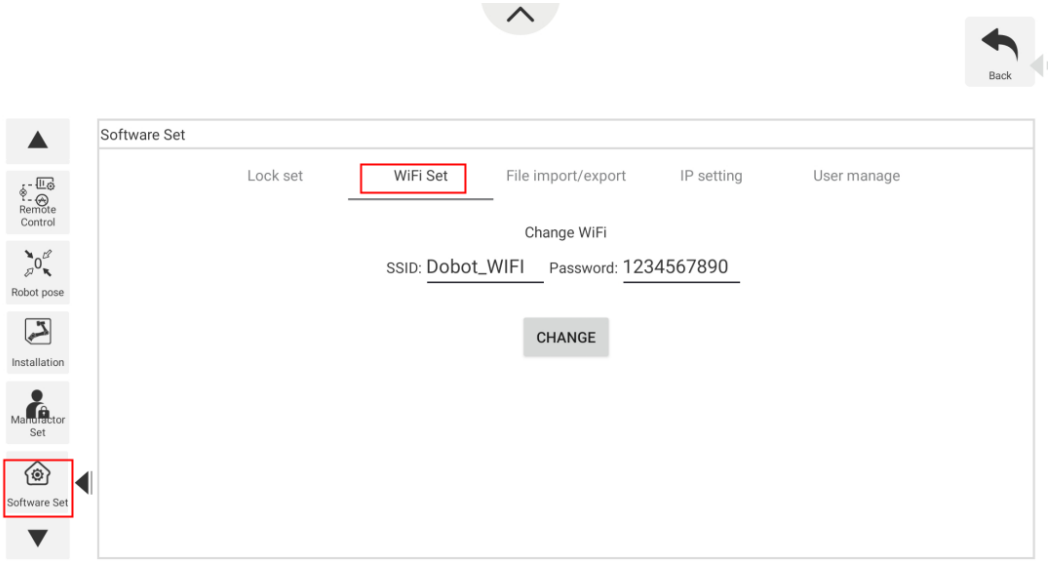

Figure 1.22 Set network

1.2.10.4 IP Setting

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 **Setting > Software Set > IP setting** page. The IP address of the robot system must be in the same network segment of the external equipment without conflict.

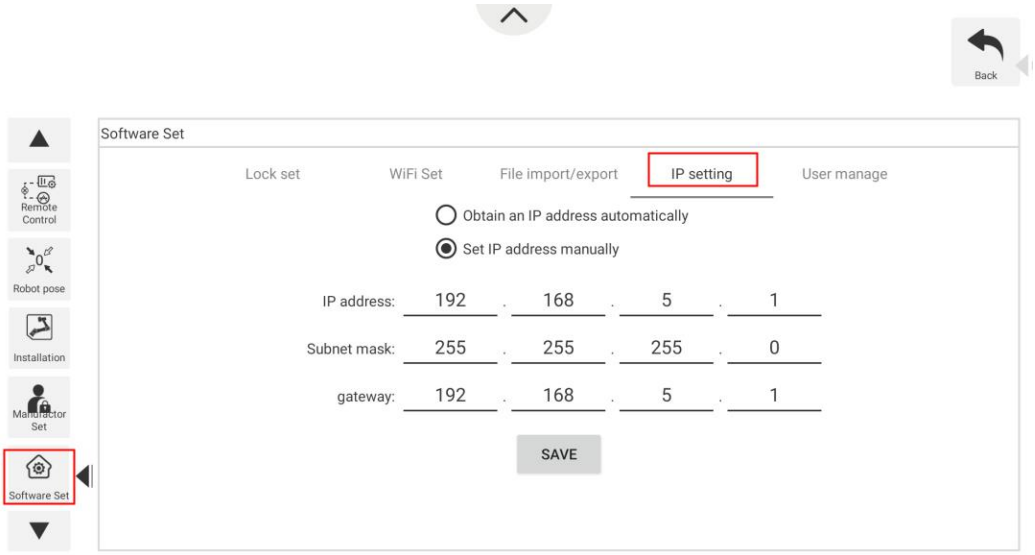

Figure 1.23 Setting controller IP

- If the robot system connects to the external equipment directly or with a switchboard, please select **Set IP address manually** 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 **Obtain an IP address automatically** to assign IP address automatically, and then click **SAVE**.

1.2.10.5 User Manager

You can modify the observer, operator, programmer, and manager's password on the User manage page.

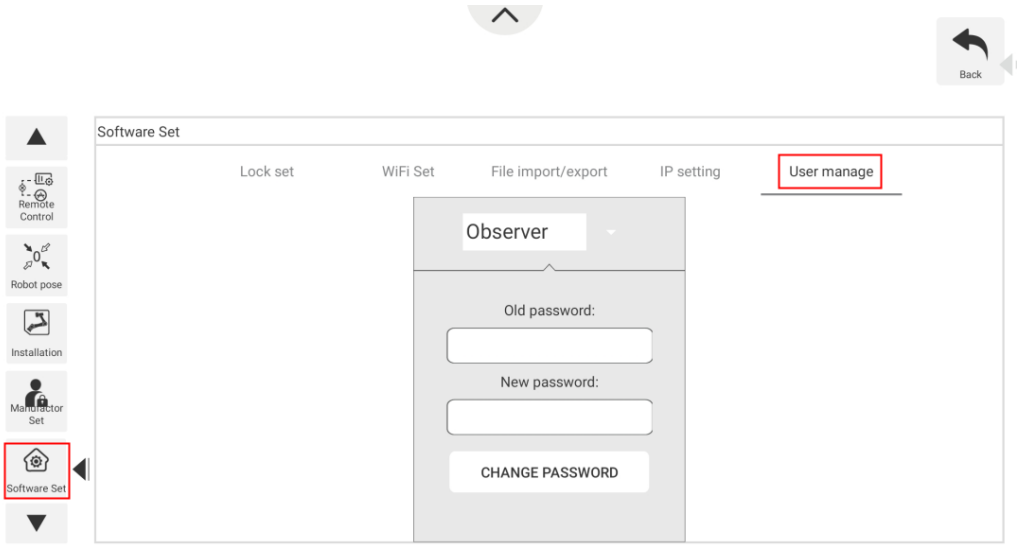

Figure 1.24 User manage

1.3 Process

This module must be operated with programmer authority or manager authority.

1.3.1 Drag Teach

You can drag robot to teach by APP or function keys on the end of robot.

Click **Craft > Drag teach**, the dragging page is displayed, as shown in Figure 1.25.

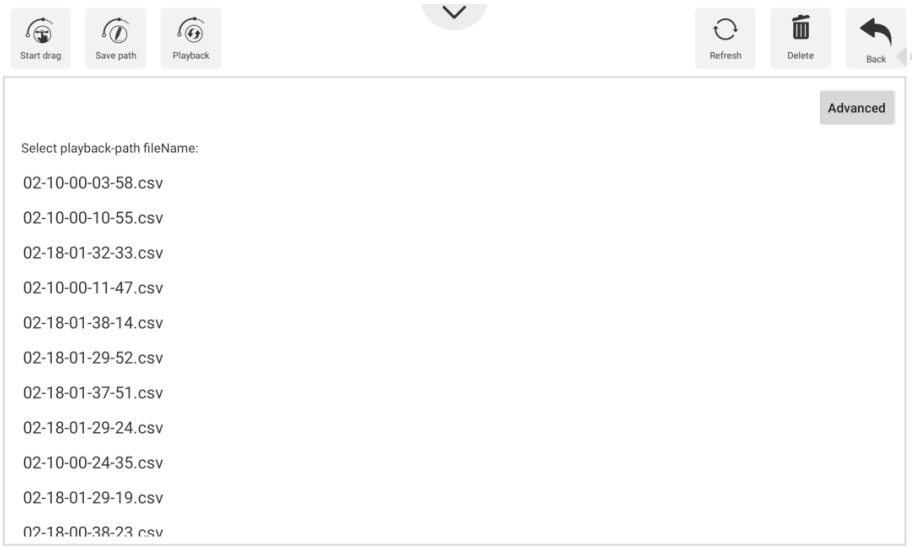

Figure 1.25 Hand-hold teach page

Table 1.4 Button description

Button	Description
	Click this button to open or close drag mode
	Click this button to save trajectory
	Click this button to start or stop playback. Select the recorded trajectory, and click Path playback to start trajectory playback
Advanced	You can set the playback loop- time, speed override and decide whether to recur the trajectory as a constant speed Speed override is that the playback speed is set to a multiple of the dragging speed, you can set to 0.25, 0.5, 1, 2, 4

📖NOTE

There are two kinds of trajectory names. For example, **01-01-00-20-07.csv** is a trajectory which records by APP. You can record multiple trajectories in this type. **Button_drag_track.csv** is a trajectory which records by clicking function keys on the end of robot. In this type, you can only record one trajectory.

1.3.2 Palletizing

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 the teach pendant, 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.

1.3.2.1 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 1.26 and Figure 1.27 show the assembly path and dismantling path.

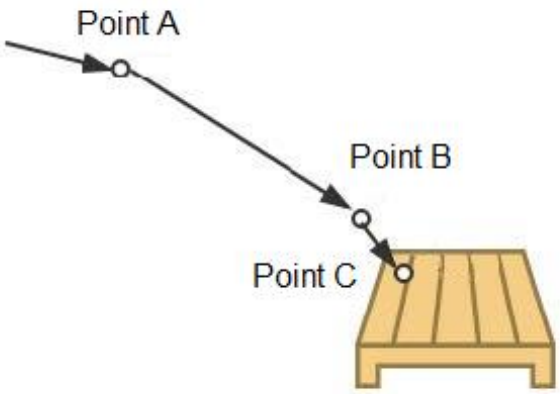

Figure 1.26 Assembly path

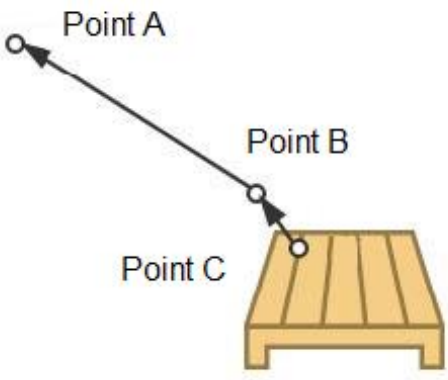

Figure 1.27 Dismantling path

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 1.28.

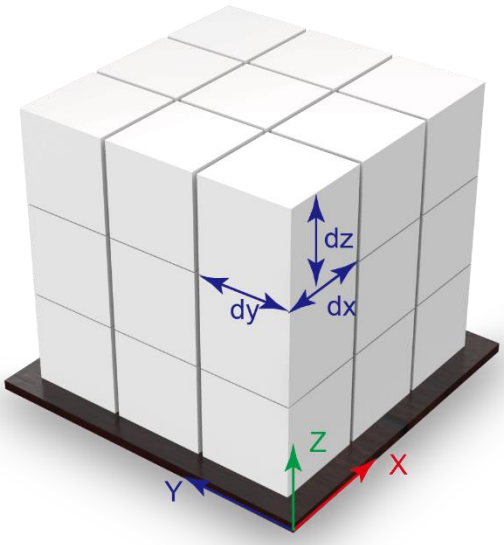

Figure 1.28 Matrix pallet

In this topic, we describe how to set pallet parameters. The 10 types of pallets can be defined.

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.
- The robot motor has been enabled.
- The user's authority is programmer authority or higher authority.

Procedure

Step 1 Click **Palletizing** on the **Craft** page.

The pallet page is displayed, as shown in Figure 1.29.

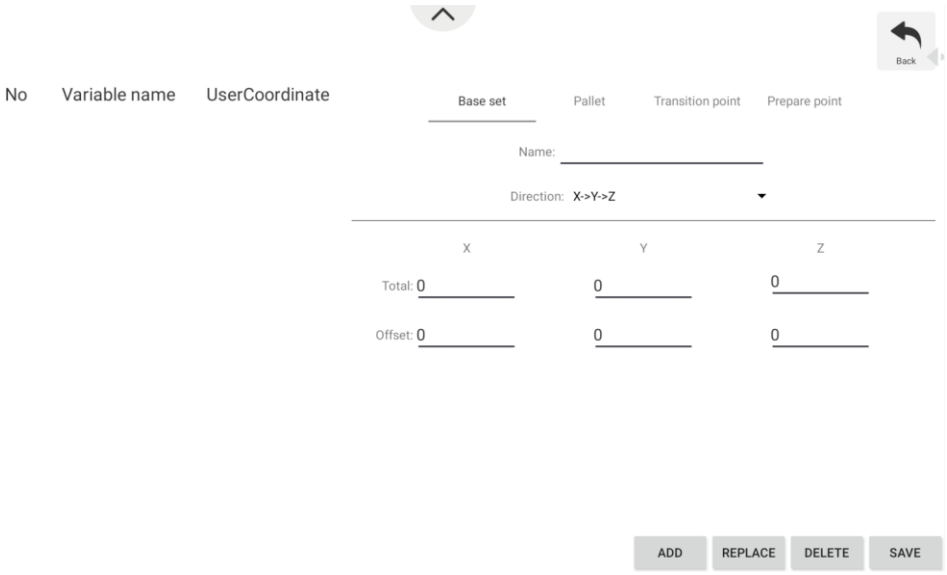

Figure 1.29 Pallet page

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

Table 1.5 shows the basic pallet parameter description.

Table 1.5 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**
Total	Number of stacks in X, Y, Z direction respectively
Offset	Stack interval in X, Y, Z direction respectively

Step 3 Jog the robot to the first stack position and click **GET COORDINATE** on the **Pallet** tab, as shown in Figure 1.30.

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

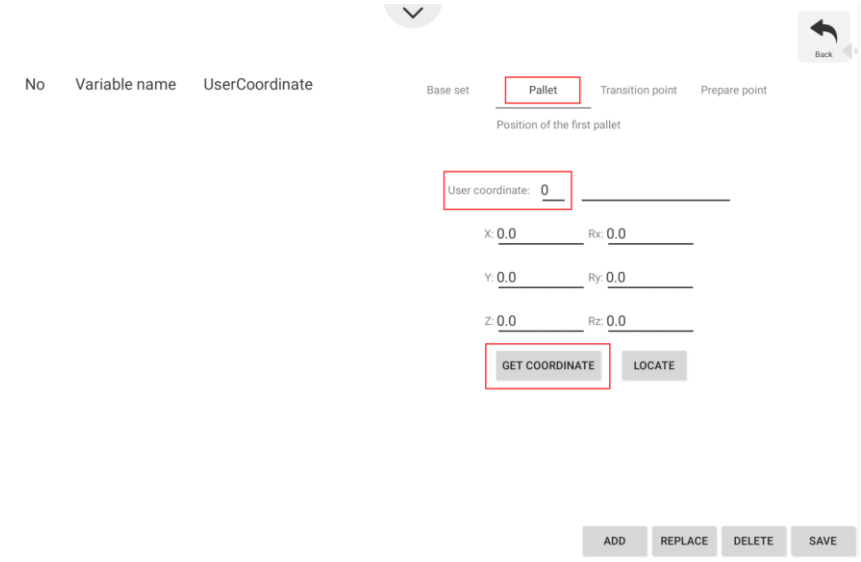

Figure 1.30 Teach the first stack position

Step 4 Jog the robot to the transition point and click **GET COORDINATE** on the **Transition Point** tab, as shown in Figure 1.31.

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

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

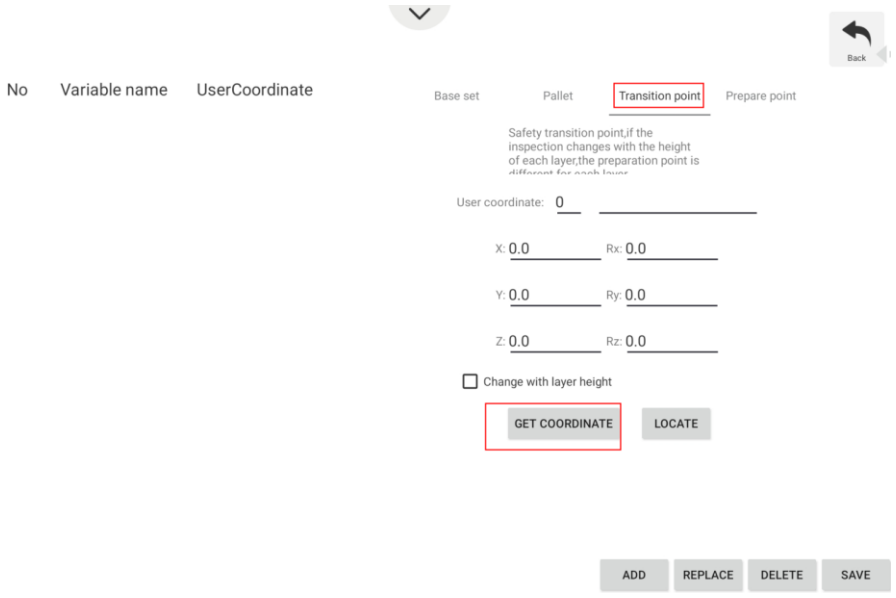

Figure 1.31 Teach the transition point

Step 5 Jog the robot to the position where is above the first stack, and click **GET COORDINATE** on the **Prepare Point** tab.

User coordinate 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.

1.3.2.2 Example

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

Program 1.1 Stack assembly demo

```
local MPpick = MatrixPallet(0,1, "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
Release(MPpick)                                                        // Release pallet instance
```

1.4 Monitor

1.4.1 I/O Monitor

You can set or monitor the I/O status of the controller and robot on this page, as shown in Figure 1.32, you can monitor I/O status with the observer authority. Also, you can set the output with the operator authority or higher authority.

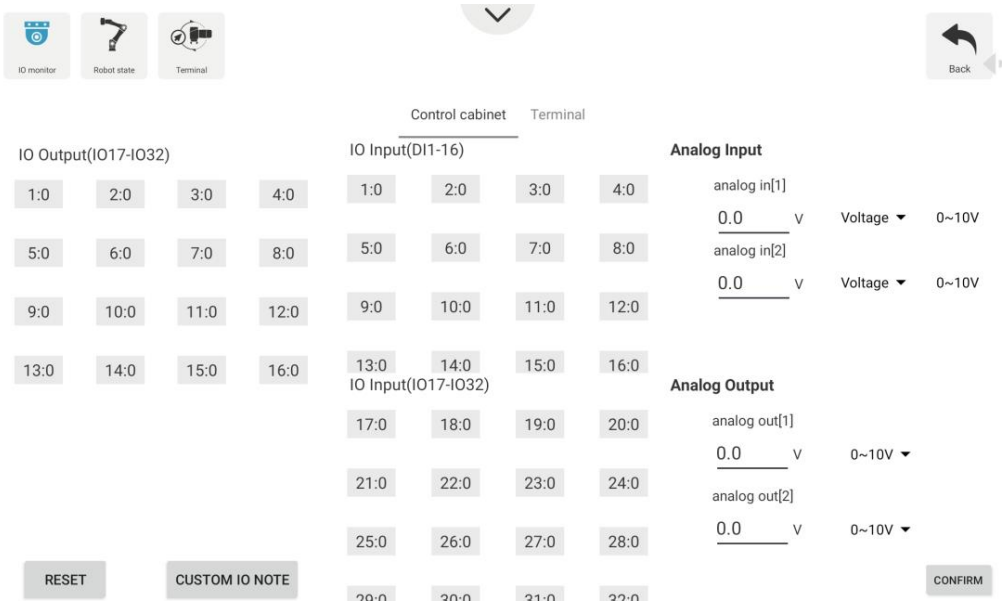

Figure 1.32 I/O Monitor

There are three features: Output, monitor and simulation.

- Output: Set the digital output or analog output with the operator authority or higher authority.
- Monitor: Check the status of the input and output with the observer authority or higher authority. The status of the output and input cannot be modified.
- Simulation: Simulate the analog input in the manual mode or auto mode for debugging and running program with the operator authority or higher authority, as shown in Figure 1.33.

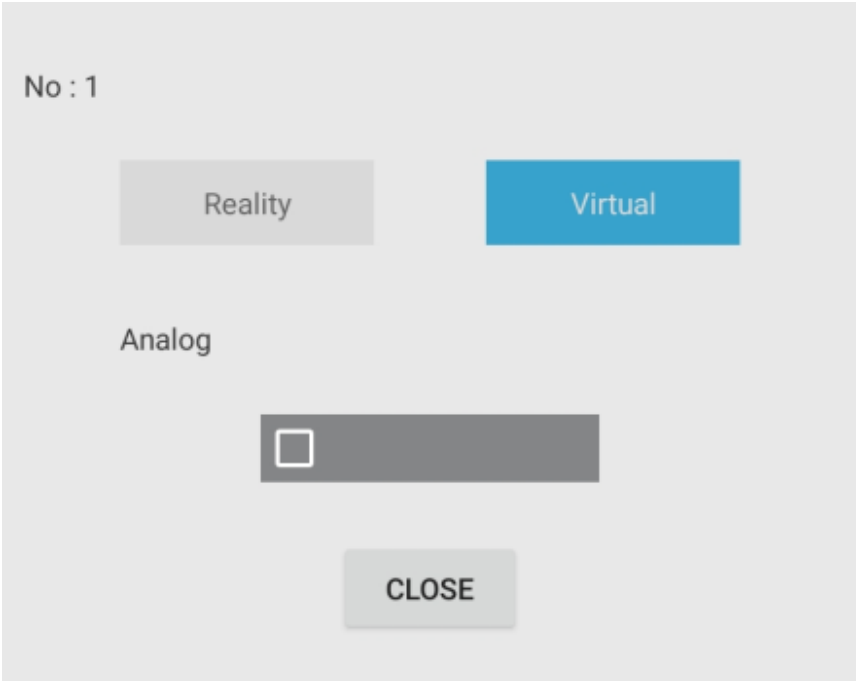

Figure 1.33 Simulation

1.4.2 Robot Status

You can check data of the robot, joint load, and running log.

Figure 1.34 Robot status

1.4.3 Terminal

This module must be operated with programmer authority or manager authority.

1.4.3.1 Custom End

You can install gripper plug-in, enable and control it on the **Monitor > Terminal > Custom end** page.

In this topic, we take DH gripper as an example to describe.

Step 1 Click  on the **Monitor > Terminal > Custom end** page to install DH gripper.

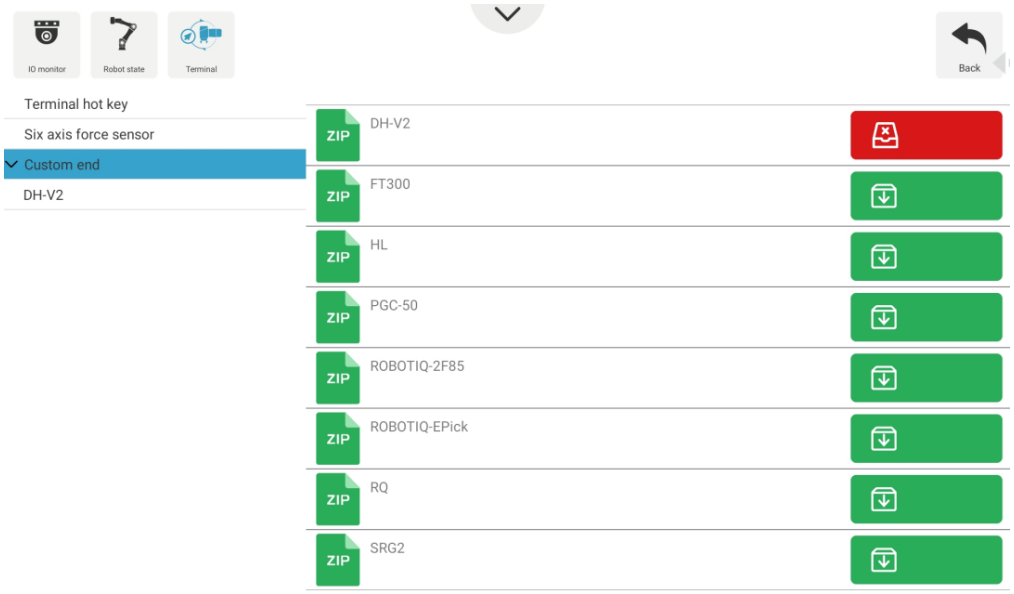

Figure 1.35 Install DH gripper

Step 2 Click **Custom end > DH-V2 > Setting**.

The **Setting** page is displayed.

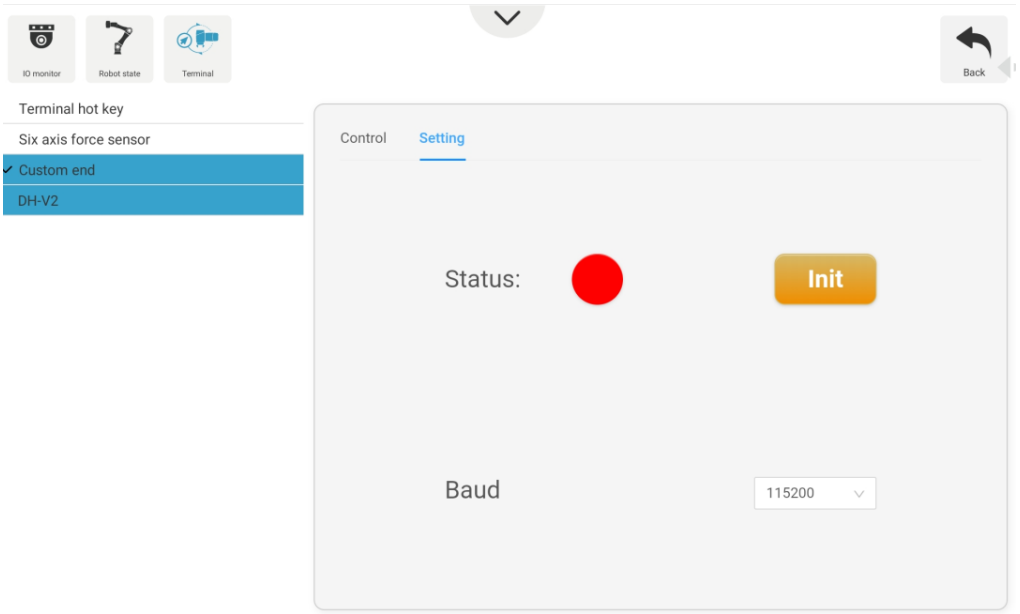

Figure 1.36 Setting page

Step 3 Set **Baud** to 115200 and click **Init** on the **Setting** page.

Step 4 Set the gripper opening and closing position and force on the Control page.

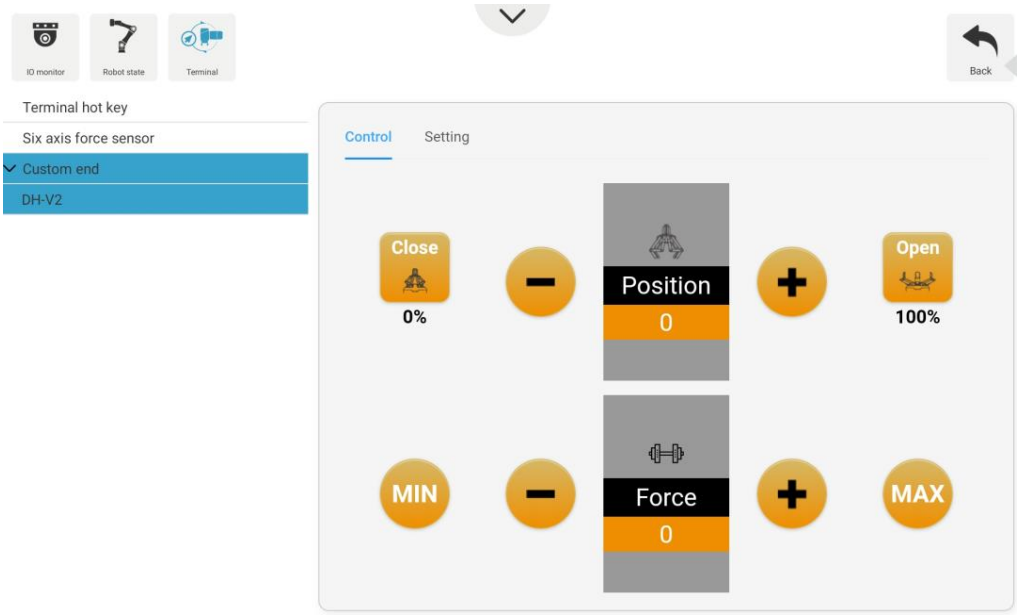

Figure 1.37 Control DH gripper

1.4.3.2 Terminal Hot Key

After installing the gripper, you can set the end-effector shortcut key on this page, so that you can control the gripper by the **control end effector** button on the end of the robot.

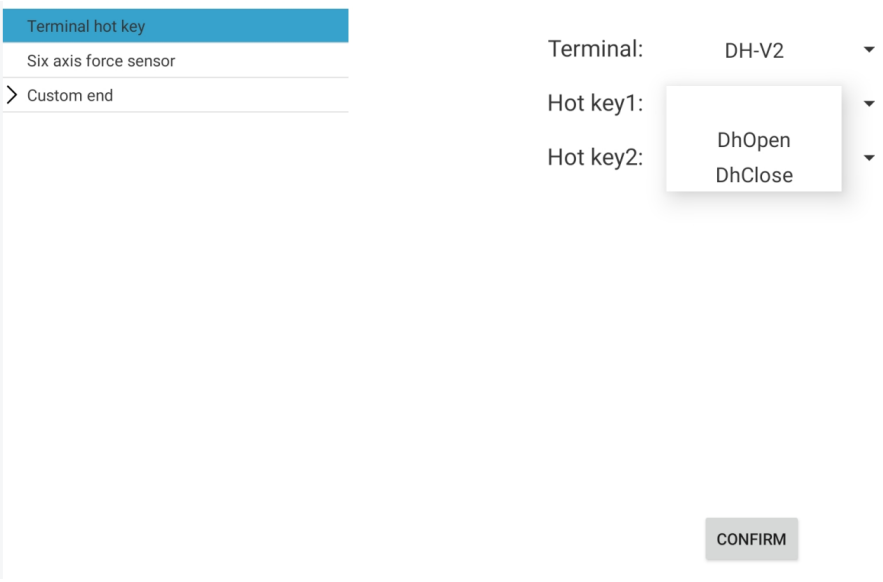

Figure 1.38 Terminal hot key

1.5 Programming

This module must be operated with the programmer authority or manager authority.

1.5.1 Project Description

The robot program is managed in project form, including teaching points list, global variables, and program files. Figure 1.39 shows the project structure.

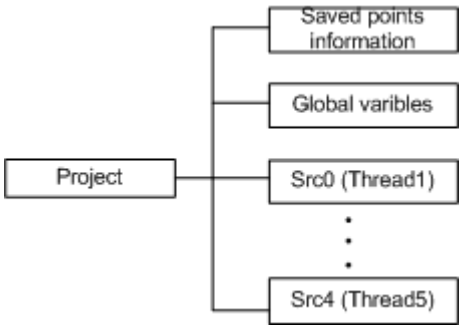

Figure 1.39 Project structure

1.5.2 Program Description

CR5 robot supports script programming and blockly programming which use the Lua language as the programming language.

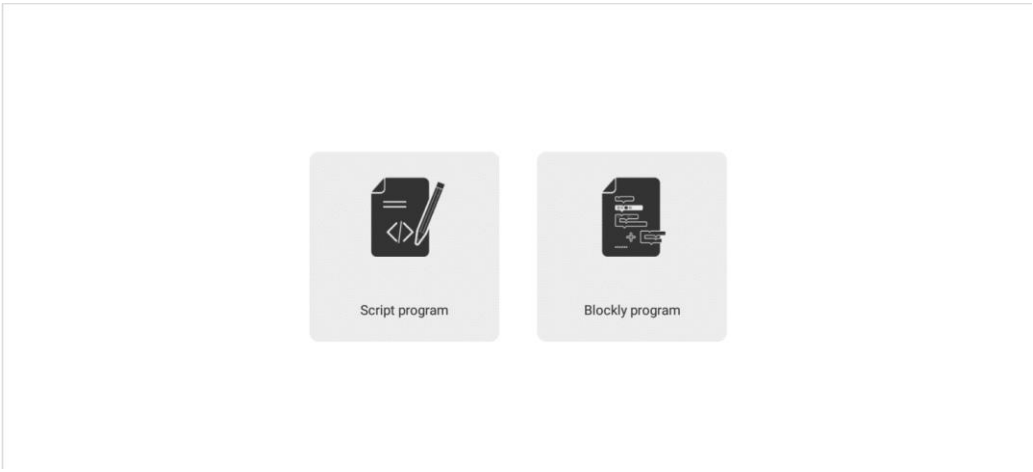

Figure 1.40 Program Description

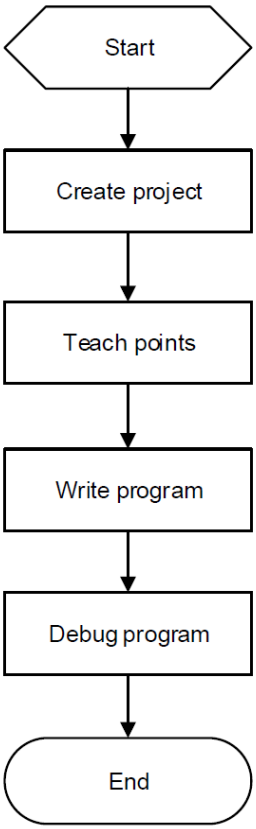

Figure 1.41 Programming process

- **Script Programming**

Script programming uses Lua as the programming language, as shown in Figure 1.42.

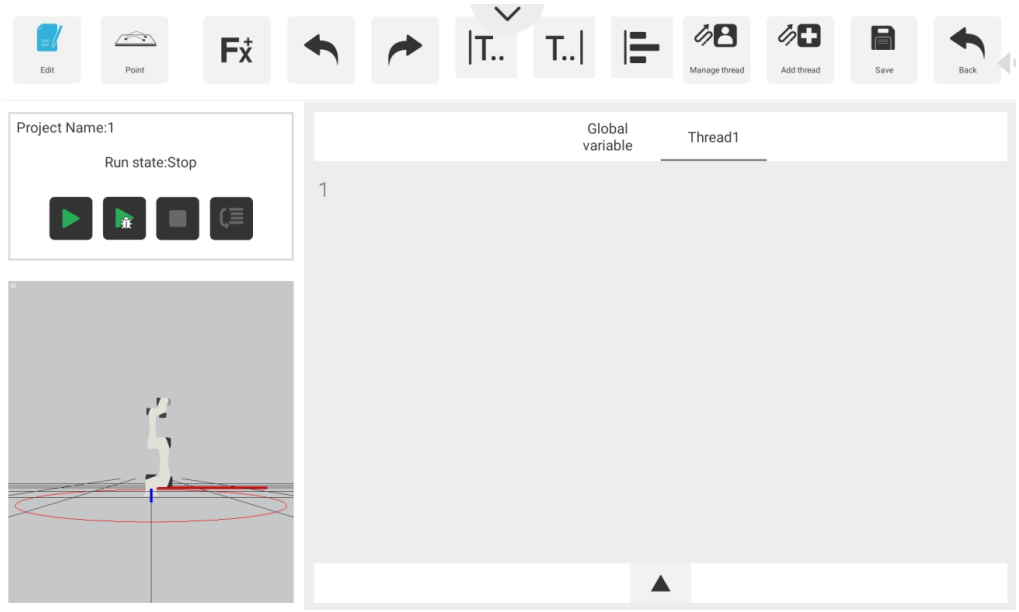

Figure 1.42 Script programming

Table 1.6 Button description

Button	Description
	Write program
	Set teaching points
	Command area, you can program with those commands
	Undo
	Recover
	Go back to the beginning of a line
	Go back to the end of a line

Button	Description
	Align code
	Delete thread
	Add thread. Up to 5 threads can be added
	Save program

- **Blockly Program**

Blockly is graphical programming. The operation of the robot can be programmed through puzzles which is intuitive and easy to understand. Only one thread is supported when using blockly.

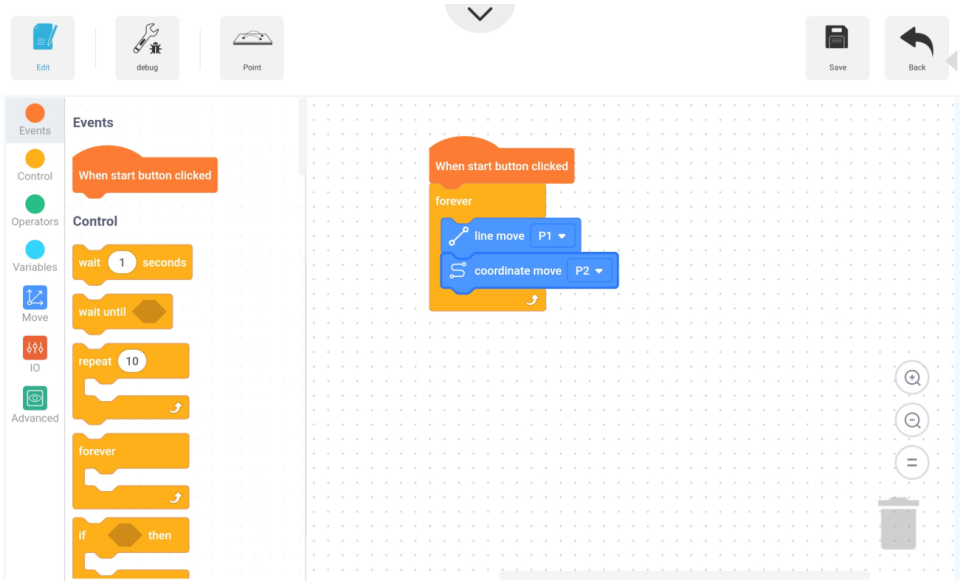

Figure 1.43 Blockly programming

1.5.3 Program Example

This section takes script programming as an example to describe how to program. We call **Go** command to make the robot move between point P1 and point P2 circularly.

Prerequisites

Robot has connected to APP.

Robot has been enabled.

Procedure

Step 1 Click **Program> Script program> New File** , select template and name this project.

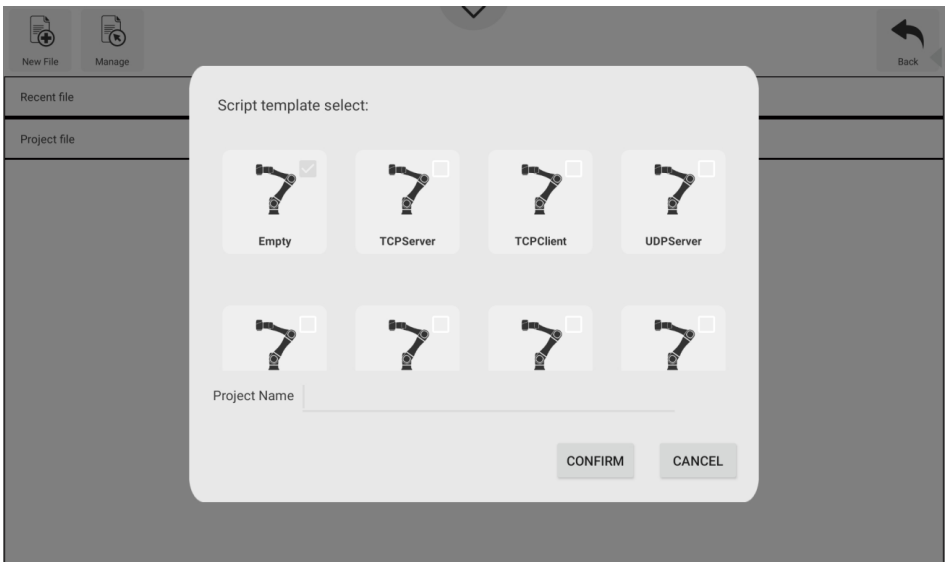

Figure 1.44 Select project template and name the project

The program page is displayed, as shown in Figure 1.45.

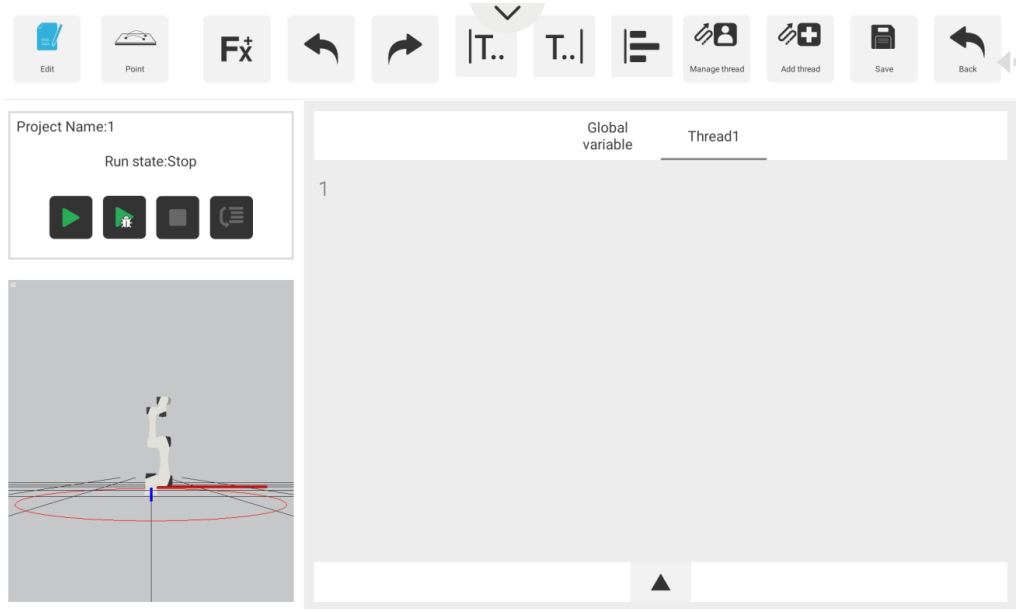

Figure 1.45 Script program page

Step 2 Click , choose **MOVE>Speed** and set speed to 40%.

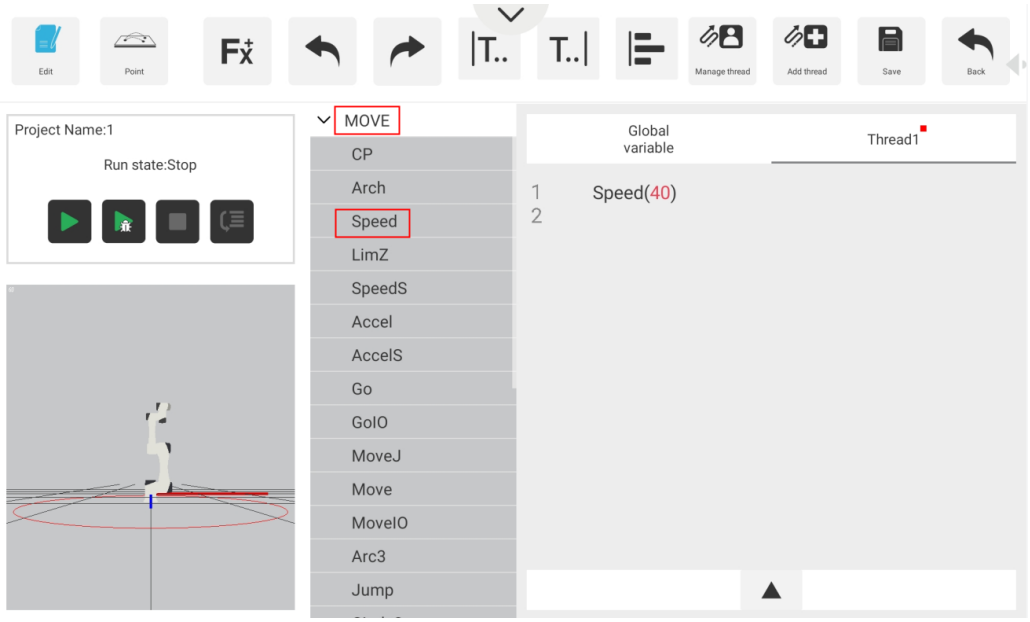

Figure 1.46 Set speed

Step 3 Jog robot to a point and click  to add the point **P1** on the **Point** page. Jog robot to another point and click  to add the point **P2**. The saved points are shown in Figure 1.47.

Arm is the arm orientation, **Tool** is tool coordinate system, **User** is User coordinate system.

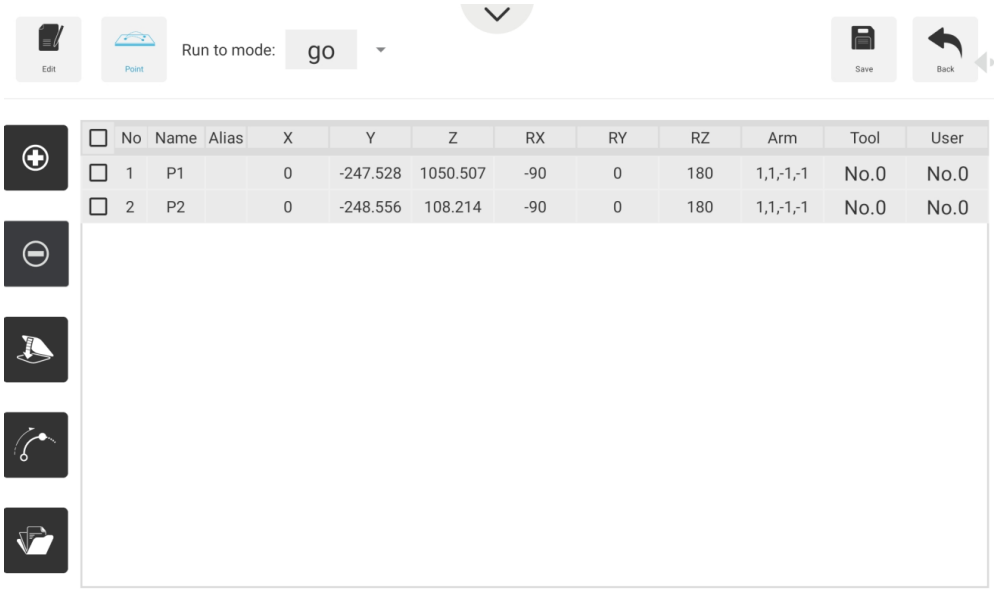

Figure 1.47 Save teaching points

Table 1.7 Button description

Button	Description
	Add a teaching point
	Delete a teaching point
	Cover a point you can select a teaching point and click **Cover** to cover the current teaching point
	Move to a point Select a point, and long press this button to make robot move to this point
	Download teaching points

Step 4 Click **Edit** to edit program, click  to select **Move>Go**, add two Go commands and select Point P1 and P2 in the two **Go** commands respectively, as shown in Figure 1.48.

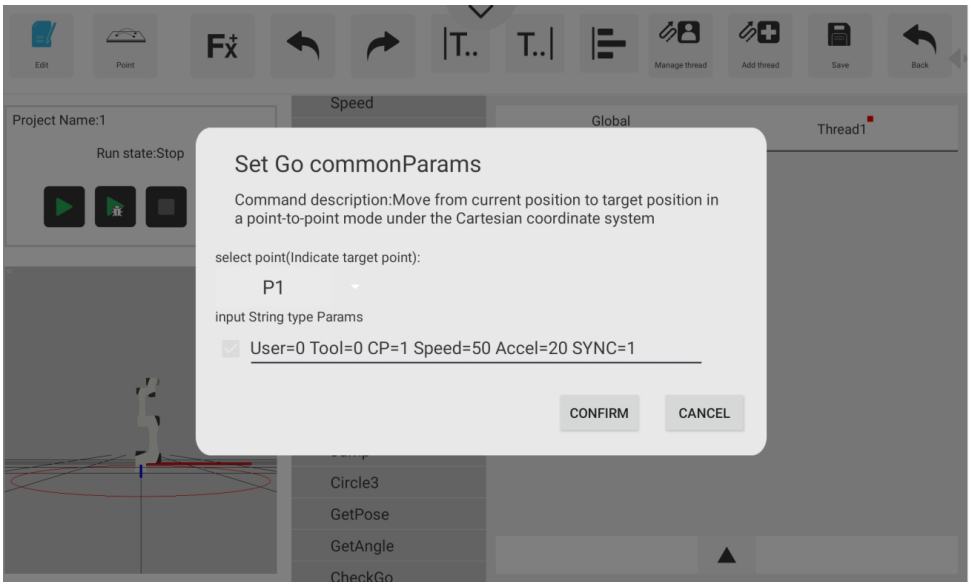

Figure 1.48 Edit program

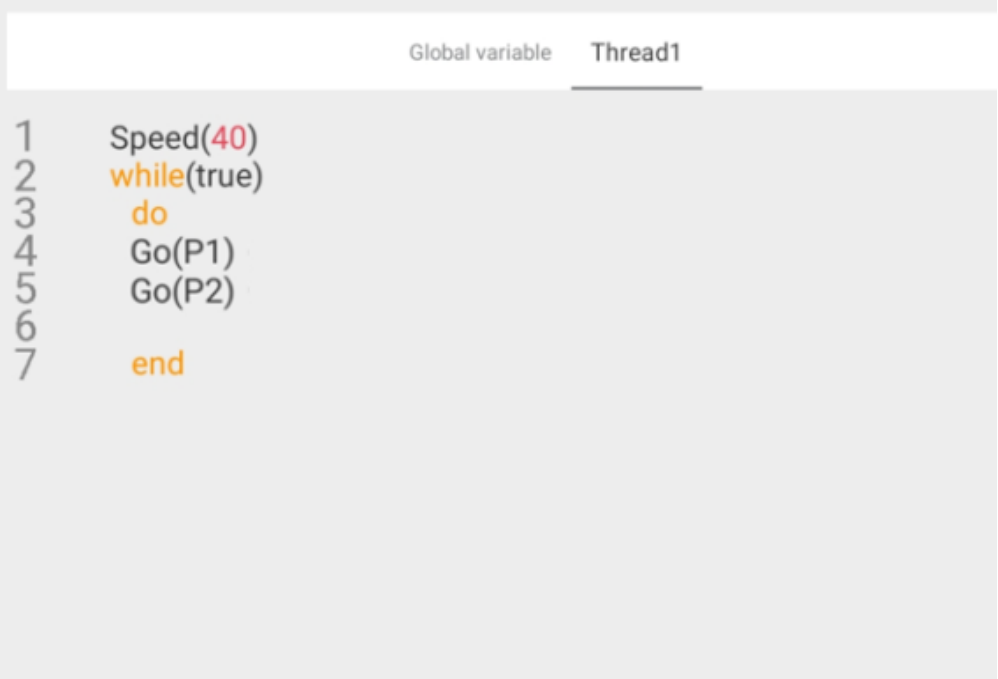

Figure 1.49 Finish programming

Step 5 Click **Save** in manual mode.

Step 6 Set running speed and click to debug this program, the robot will move between point **P1** and point **P2**. Also, you can click to stop it.

Table 1.8 Button description

Button	Description
	Start button, click it to run program
	Debug button 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
	Pause button, click it to pause program
	Run a program step by step, you can run a program step by step by setting a breakpoint

Button	Description
	This button is valid only if  turns into 

2. Program Language

CC 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
```

- 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	Robot moves from the current position to a target position in the **Move** mode. The trajectory looks like a door
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

Command	Description
	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	Go(P," User=*1* Tool=*2* CP=*1* Speed=*50* Accel=*20* *SYNC*=1")
Description	Move from the current position to a target position in a point-to-point mode under the Cartesian coordinate system
Parameter	Required parameter: P: Indicate target point, which is user-defined or obtained from the **TeachPoint** page. Only Cartesian coordinate points are supported Optional parameter: • 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	The robot moves to point P1 as the default setting Go(P1)

Table 2.9 MoveJ command

Function	MoveJ(P," CP=*1* Speed=*50* Accel=*20* *SYNC*=1")
Description	Move from the current position to a target position in a point-to-point motion under the Joint

	coordinate system
Parameter	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 Optional parameter: • 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	local P = {joint={0,-0.0674194,0,0}} MoveJ(P)

Table 2.10 Move command

Function	Move(P,"User=*1* Tool=*2* CP=*1* SpeedS=*50* AccelS=*20* *SYNC*=1")
Description	Move from the current position to a target position in a straight line under the Cartesian coordinate system
Parameter	Required parameter: P: Indicate the target point, which is user-defined or obtained from the **TeachPoint** page. Only Cartesian coordinate points are supported Optional parameter: • 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	The robot moves to point P1 as the default setting Move(P1)

Table 2.11 Arc3 command

Function	Arc3(P1,P2, " User=*1* Tool=*2* CP=*1* SpeedS=*50* AccelS=*20* *SYNC*=1")

Description	Move from the current position to a target position in an arc interpolated mode under the Cartesian coordinate system This command needs to combine with other motion commands, to obtain the starting point of an arc trajectory
Parameter	Required parameter: • 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 Optional parameter: • 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	While true do Go(P1) Arc3(P2,P3) end The robot cycles from point P1 to point P3 in the arc interpolated mode

Table 2.12 Jump command

Function	Jump(P," User=*1* Tool=*2* SpeedS=*50* AccelS=*20* Start=*10* Zlimit=*80* End=*50* *SYNC*=1")
Description	The robot moves from the current position to a target position in the **Move** mode. The trajectory looks like a door
Parameter	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 Optional parameter: • User: Indicate User coordinate system. Value range: 0 - 9 • Tool: Indicate Tool coordinate system. Value range: 0 - 9 • SpeedS: Velocity rate. Value range: 1 - 100

	• AccelS: 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	Jump(P1) The robot moves to point P1 in the Jump mode

⚠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, " User=*1* Tool=*2* CP=*1*SpeedS=*50* AccelS=*20*")
Description	Move from the current position to a target position in a circular interpolated mode under the Cartesian coordinate system This command needs to combine with other motion commands, to obtain the starting point of an arc trajectory
Parameter	Required parameter • 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 • Count: Number of circles. Value range: 1 - 999 Optional parameter: • 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) Circle3(P2,P3,1) Robot cycles from point P1 to point P3 in the circular interpolated mode

Table 2.14 RP command

Function	RP(P1, {OffsetX, OffsetY, OffsetZ})
Description	Set the X, Y, Z axes offset under the Cartesian coordinate system to return a new Cartesian coordinate point The robot can move to this point in all motion commands except MoveJ
Parameter	• 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	P2=RP(P1, {50,10,32}) Move(P2) or Move(RP(P1, {50,10,32}))

Table 2.15 RJ command

Function	RJ(P1, {Offset1, Offset2, Offset3, Offset4, Offset5, Offset6})
Description	Set the joint offset in the Joint coordinate system to return a new joint coordinate point The robot can move to this point only in **MoveJ** command
Parameter	• 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. Unit: °
Return	Joint coordinate point
Example	local P1 = {joint={0,-0.0674194,0,0}} P2=RJ(P1, {60,50,32,30}) MoveJ(P2) or MoveJ(RJ(P1, {60,50,32,30}))

Table 2.16 GoR command

Function	GoR({OffsetX, OffsetY, OffsetZ},” User=*1* Tool=*2* CP=*1* Speed=*50* Accel=*20* *SYNC*=1 ”)
Description	Move from the current position to the offset position in a point-to-point mode under the Cartesian coordinate system
Parameter	Required parameter: OffsetX, OffsetY, OffsetZ: X, Y, Z axes offset in the Cartesian coordinate system Unit: mm Optional parameter: • 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	Go(P1) GoR({10,10,10},"Accel=100 Speed=100 CP=100")

Table 2.17 MoveJR command

Function	MoveJR({Offset1, Offset2, Offset3, Offset4, Offset5, Offset6},” CP=*1* Speed=*50* Accel=*20* *SYNC*=1”)
Description	Move from the current position to the offset position in a point-to-point motion under the Joint coordinate system
Parameter	Required parameter: Offset1 - Offset6: J1 - J6 axes offset. Unit: ° Optional parameter: • 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	Go(P1)

	MoveJR({20,20,10,0},"SYNC=1")

Table 2.18 MoveR command

Function	MoveR({OffsetX, OffsetY, OffsetZ},” User=*1* Tool=*2* CP=*1* SpeedS=*50* AccelS=*20* *SYNC*=1”)
Description	Move from the current position to the offset position in a straight line under the Cartesian coordinate system
Parameter	Required parameter: OffsetX, OffsetY, OffsetZ: X, Y, Z axes offset in the Cartesian coordinate system Unit: mm Optional parameter: • 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**, **Jump**, **Arc3**, or **Circle3**
Speed	Set the velocity rate. This command is valid only when the motion mode is **Go**, or **MoveJ**
SpeedS	Set the velocity rate. This command is valid only when the motion mode is **Move**, **Jump**, **Arc3**, or **Circle3**

Command	Description
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	AccelS(20) Move(P1) The robot moves to point P1 with 20% acceleration rate

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	Speed(20) Go(P1) The robot moves to point P1 with 20% velocity rate

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	SpeedS(20) Move(P1) The robot moves to point P1 with 20% velocity rate

Table 2.24 Arch command

Function	Arch(Index)
Description	Set the index of sets of parameters (**StartHeight**, **zLimit**, **EndHeight**) in the **Jump** mode
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 **Setting** > **PlaybackArch** of the APP
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

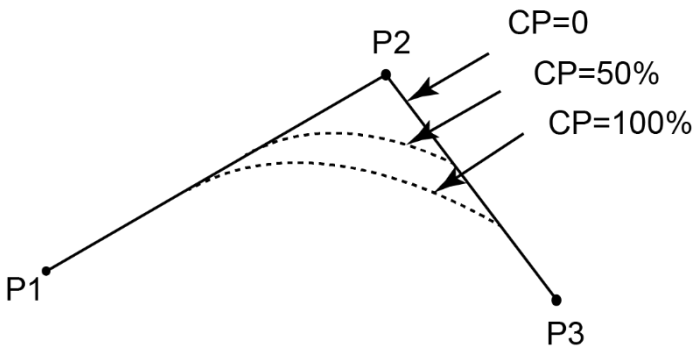

Figure 2.1 Continuous path

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 Six-axis Force Sensor Commands

Table 2.27 Six-axis force sensor command

Command	Description
SixForceHome	Six-axis force sensor homing command
Spiral	Six-axis force sensor spiral motion command
Rotation	Six-axis force sensor rotation motion command
Linear	Six-axis force sensor linear motion command

Table 2.28 Six-axis force sensor homing command

Function	SixForceHome
Description	Homing six-axis force sensor
Parameter	None
Example	SixForceHome(): execute the command to home six-axis force sensor

Table 2.29 Six-axis force sensor spiral command

Function	Spiral(P, User, Tool, Direction, SpeedC, Force, Insertion, Perturn, PeckMode, MaxValue)
Description	The robot arm performs a spiral motion between the current position and the specified position to find the hole position. The specified point needs to be closer to the hole position, which is the starting point for hole position exploration.
Parameter	• P: the specified position • User: User coordinate system, value range: 0~9 • Tool: Tool coordinate system, Value range: 0~9 • Direction: Jack direction (0: Forward, 1: Reverse) • SpeedC: Jack speed(mm/s) • Force: Spiral threshold (N) • Insertion: Jack threshold (N) • Perturn: Spiral radius (mm) • PeckMode: Point contact mode (ON/OFF) • MaxValue: Maximum spiral radius (mm)
Example	Spiral(P1,User=1 Tool=2 Dirction=0 SpeedC=5 Force =10 Insertion=3 Perturn=0.7 PeckMode=OFF MaxValue =5") Do a spiral motion between the current position and P1 to find the hole position. When the resistance in the direction of the jack is greater than the Force threshold, the robot performs a spiral motion to explore the hole position. When the resistance in the direction of the jack is less than the Insertion threshold, the robot moves in the direction of the jack to perform the jack work.

Table 2.30 Six-axis force sensor rotation command

Function	Rotation (P, User, Tool, Direction, SpeedC, Force, RotationSpeed, MaxTorque, PeckMode, MaxValue)
Description	The robotic arm rotates between the current position and the specified position to find the hole position. The specified point needs to be close to the hole position, which is the starting point for hole position exploration.

Parameter	• P: the specified position • User: User coordinate system, value range: 0~9 • Tool: Tool coordinate system, Value range: 0~9 • Direction: Jack direction (0: Forward, 1: Reverse) • SpeedC: Jack speed(mm/s) • Force: Rotation threshold (N) • RotationSpeed: Rotation speed (%s) • MaxTorque: Maximum torque (Nm) • PeckMode: Point contact mode (ON/OFF) • MaxValue: Maximum spiral radius (mm)
Example	Rotation (P1, "User=1 Tool=2 Dirction=0 SpeedC =5 Force =10 RotationSpeed=5 MaxTorque=1 PeckMode=OFF MaxValue =45") Do a rotation motion between the current position and P1 to find the hole position. When the resistance in the direction of the jack is greater than the Force threshold, the robot performs a rotation motion to explore the hole position. When the resistance in the direction of the jack is less than the Force threshold, the robot moves in the direction of the jack to perform the jack work.

Table 2.31 Six-axis force sensor linear command

Function	Linear (User, Tool, Direction, SpeedC, Force, MaxValue)
Description	The robot arm makes a linear jack movement in the direction of the hole
Parameter	• User: User coordinate system, value range: 0~9 • Tool: Tool coordinate system, Value range: 0~9 • Direction: Jack direction (0: Forward, 1: Reverse) • SpeedC: Jack speed(mm/s) • Force: Rotation threshold (N) • MaxValue: Maximum spiral radius (mm)
Example	Linear("User=1 Tool=2 Dirction=0 SpeedC =5 Force=10 MaxValue=45") Do a linear jack movement at the current hole position. When the resistance in the insertion direction is greater than the Force threshold, the insertion is considered complete.

2.11 Input/output Commands

Table 2.32 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 2.33 Digital input command

Function	DI(*index*)
Description	Get the status of the digital input port
Parameter	index: Digital input index. Value range: 1 - 16
Return	• When an index is set in the DI function, **DI(index)** returns the status (ON/OFF) of this speicified 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 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]**
Example	if (DI(1))==ON then Move(P1) end The robot moves to point P1 when the status of the digital input port **1** is **ON**

Table 2.34 Digital output command (Queue command)

Function	DO(*index, ON | OFF*)

Description	Set the status of digital output port (Queue command)
Parameter	• index: Digital output index. Value range: 1- 24 • ON/OFF: Status of the digital output port. ON: High level; OFF: Low level
Example	DO(1,ON) Set the status of the digital output port 1to **ON**

Table 2.35 Digital output command (Immediate command)

Function	DOExecute(*index, ON | OFF*)
Description	Set the status of digital output port (Immediate command)
Parameter	• index: Digital output index. Value range: 1 - 24 • ON/OFF: Status of the digital output port. ON: High level; OFF: Low level
Example	DOExecute(1,OFF) Set the status of the digital output port 1 to **OFF**

2.12 Program Managing Commands

Table 2.36 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 time
ElapsedTime	Stop time
Systime	Get the current time

Table 2.37 Wait command

Function	Wait(*time*)
Description	Set the delay time for robot motion commands
Parameter	time: Delay time. Unit: ms
Example	Go(P1) Wait(1000) Wait for 1000ms after the robot moves to point P1

Table 2.38 Sleep command

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

Table 2.39 Pause command

Function	Pause()
Description	Pause the running program When the program runs to this command, robot pauses running and the button  on the APP turns into  . If the robot continues to run, please click 
Parameter	None
Example	while true do Go(P1) Go(P2) Pause() Go(P3) Go(P4) end The robot moves to point P2 and then pauses running

Table 2.40 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	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)

Table 2.41 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	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)

Table 2.42 Get current time command

Function	Systime()

Description	Get the current time
Parameter	None
Return	Current time
Example	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)

2.13 Pose Getting Command

Table 2.43 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	local currentPose = GetPose() --Get the current pose local liftPose = {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)

Table 2.44 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	local armPose local joint = GetAngle() --Get the current pose local liftPose = {armOrientation = armPose , joint = {joint.joint[1], joint.joint[2], joint.joint[3], joint.joint[4]}, tool = 0, user = 0}

2.14 TCP

Table 2.45 Create TCP command

Function	err, socket = TCPCreate(*isServer*, *IP*, *port*)
Description	Create a TCP network Only support a single connection
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 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
Return	err: 0: TCP network is created successfully 1: TCP network is created failed Socket: Socket object
Example	Please refer to Program 2.1 and Program 2.2

Table 2.46 TCP connection command

Function	TCPStart(*socket*, *timeout*)
Description	Connect a client to a server with the TCP protocol
Parameter	socket: Socket object timeout: Wait timeout. Unit: s. If **timeout** is 0, the connection is still waiting. If not, after exceeding the timeout, the connection is exited.

Return	• 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 2.47 Receive data command

Function	err, Recbuf = TCPRead(*socket*, *timeout*, *type*)
Description	Robot as a client receives data from a server Robot as a server receives data from a client
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 2.48 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 2.49 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.

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(buf)
```

```
                else
                    print("Read error ".. err)
                    break
                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(buf)
                else
                    print("Read error ".. err)
                    break
                end
            end
```

```
    else
        print("Create failed ".. err)
    end
    TCPDestroy(socket)
else
    print("Create failed ".. err)
end
```

2.15 UDP

Table 2.50 Create UDP network command

Function	err, socket = UDPCreate(*isServer*, *IP*, *port*)
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 2.51 Receive data command

Function	err, Recbuf = UDPRead(*socket*, *timeout*, *type*)
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 2.52 Send data command

Function	UDPWrite(*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 2.3 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.

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
                Go(P2)
                print(buf)
            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(buf)
            else
                print("Read error ".. err)
                break
          end
```

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

2.16 Modbus

2.16.1 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 salve.

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 2.53 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 2.54 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

Discrete input register address (e.g.: PLC)	Discrete input register address(Robot system)	Data type	Description
10004	3	Bit	Running state
10005	4	Bit	Alarm state
10006-10999	5-998	Bit	Reserved
11000-14096	999-4095	Bit	User-defined

- Input register

Table 2.55 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 2.56 Holding register description

Holding register address (e.g.: PLC)	Holding register address (Robot system)	Data type	Description
40001-41000	0-999	Byte	Reserved
41001-44096	1000-4095	Byte	User-defined

2.16.2 Command Description

Table 2.57 Rea coil register command

Function	GetCoils(*addr*, *count*)
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 Coils = GetCoils(0,5) Return: Coils={1,0,0,0,0} As shown in Table 2.52, it indicates that the robot is in the starting state

Table 2.58 Set coil register command

Function	SetCoils(*addr*, *count*, *table*)
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 2.59 Read discrete input register command

Function	GetInBits(*addr*, *count*)
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 2.53, it indicates the robot is in running state

Table 2.60 Read input register command

Function	GetinRegs(*addr*, *count*, *type*)
Description	Read the input register value with the specified data type from the Modbus slave
Parameter	addr: Starting address of the input registers. Value range: 0 - 4095 count: Number of the input registers to read. Value range: 0 ~ 4096-addr type: Data type • 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 data = GetInRegs(2048,1) Example 2: Read a 32-bit unsigned integer starting at address 2048 data = GetInRegs(2048, 1, “U32”)

Table 2.61 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	addr: Starting address of the holding registers. Value range: 0 - 4095 count: Number of the holding registers to read. Value range: 0 to 4096-addr type: Datatype • 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 data = GetHoldRegs(2048,1) Example 1: Read a 32-bit unsigned integer starting at address 2048 data = GetInRegs(2048, 1, “U32”)

Table 2.62 Set holding register command

Function	SetHoldRegs(*addr*, *count*, *table*, *type*)
Description	Set the holding register in the Modbus slave
Parameter	addr: Starting address of the holding registers to set. Value range: 0 - 4095 count: Number of the holding registers to set. Value range: 0 to 4096-addr table: Holding register value, stored in a table type: Datatype • 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	Example1: Set a 16-bit unsigned integer starting at address 2048 local data = {6000} SetHoldRegs(2048, #data, data, “U16”) Example2: Set a 64-bit double-precision floating-point number starting at address 2048 local data = {95.32105} SetHoldRegs(2048, #data, data, “F64”)

2.17 Process Command

2.17.1 Pallet Commands

Table 2.63 Create matrix pallet command

Function	Pallet = MatrixPallet (*index*, *ID*, “IsUnstack= *true* Userframe= *1*”)
Description	Instantiate matrix pallet

Parameter	Index: Matrix pallet index ID: Unique identification of pallet 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	myPallet = MatrixPallet(0,1,“IsUnstack=tsrue Userframe=8”)

Table 2.64 Set the next stack index command

Function	SetPartIndex (Pallet, index)
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	local myPallet = MatrixPallet(0,1, “IsUnstack=true Userframe=8”) SetPartIndex(myPallet,1) The next stack index to be operated is 2

Table 2.65 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	local index=GetPartIndex(myPallet) If the return value is 1, it indicates that the current operated stack index is 2

Table 2.66 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	local myPallet = MatrixPallet(0,1, “IsUnstack=true Userframe=8”) SetPartIndex(myPallet,1) The next pallet layer index to be operated is 2

Table 2.67 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	local index=GetLayerIndex(myPallet) If the return value is 1, it indicates that the current operated pallet layer index is 2

Table 2.68 Reset command

Function	Restet (Pallet)
Description	Reset pallet
Parameter	Pallet: Pallet object
Return	None
Example	local myPallet = MatrixPallet(0,1, “IsUnstack=true Userframe=8”) Reset(myPallet)

Table 2.69 Check the pallet status command

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

Table 2.70 Release pallet command

Function	Release (Pallet)
Description	Release pallet object
Parameter	Pallet: Pallet object
Return	None
Example	Release(myPallet)

Table 2.71 MoveIn command

Function	MoveIn (Pallet, "velAB=*20* velBC=*30* accAB=*20* accBC=*10* CP=*20* SYNC=*1*")
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: • 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 2.72 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 • 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 2.2 and Figure 2.3 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.

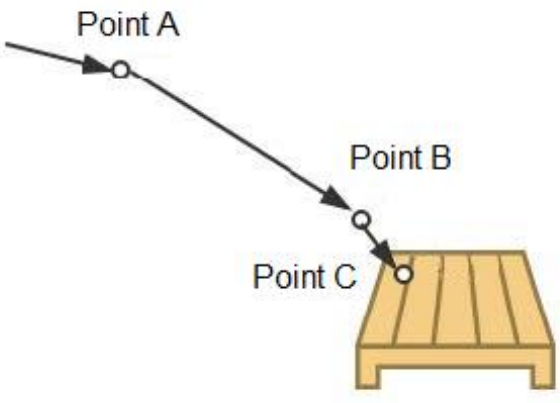

Figure 2.2 Stack assembly path

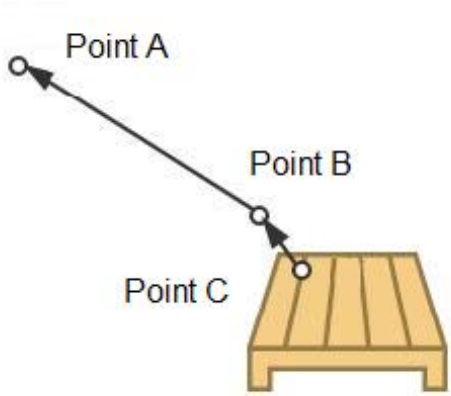

Figure 2.3 Stack dismantling path

