Dobot CR Series Robot Program Guide

Issue: V1.0

Date: 2020-12-25

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 robot API commands for programming with Lua language..

Intended Audience

This document is intended for:

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

Change History

Date	Change Description
2020/12/25	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. Overview

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

2. 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

3. Relational Operator

Table 3.1 Relational Operator

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

4. Logical Operators

Table 4.1 Logical operator

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

5. General Keywords

Table 5.1 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

6. General Symbol

Table 6.1 General symbol

Command	Description
#	Get the length of the array **table**

7. Processing Control Commands

Table 7.1 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)

8. 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
```


9. Motion Commands

Table 9.1 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 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 9.2 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 9.3 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 9.4 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 9.5 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 9.6 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 9.7 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 9.8 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 9.9 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 9.10 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 9.11 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 9.12 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")

10. Motion Parameter Commands

Table 10.1 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**
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 10.2 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 10.3 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)

Function	AccelS(R)
	Move(P1) The robot moves to point P1 with 20% acceleration rate

Table 10.4 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 10.5 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 10.6 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 10.7 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

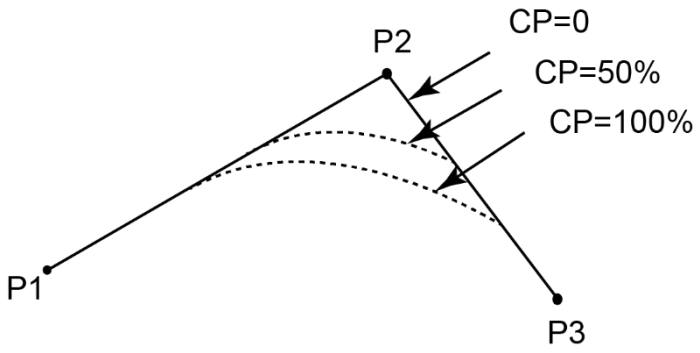

Figure 10.1 Continuous path

Table 10.8 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 ")

11. Six-axis Force Sensor Commands

Table 11.1 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 11.2 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 11.3 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 11.4 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 11.5 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.

12. Input/output Commands

Table 12.1 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 12.2 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 12.3 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 12.4 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**

13. Program Managing Commands

Table 13.1 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 13.2 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 13.3 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 13.4 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 13.5 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 13.6 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 13.7 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)

14. Pose Getting Command

Table 14.1 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 14.2 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}

15. TCP

Table 15.1 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 15.1 and Program 15.2

Table 15.2 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 15.1 and Program 15.2

Table 15.3 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 15.1 and Program 15.2

Table 15.4 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 15.1 and Program 15.2

Table 15.5 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 15.1 and Program 15.2

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 15.1 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 15.2 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
```

16. UDP

Table 16.1 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 16.1 and Program 16.2

Table 16.2 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 16.1 and Program 16.2

Table 16.3 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 16.1 and Program 16.2

⚠ 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 16.1 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 16.2 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
```


17. Modbus

17.1 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 17.1 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 17.2 Discrete input register description

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

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

- Input register

Table 17.3 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 17.4 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

17.2 Command Description

Table 17.5 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 16.3, it indicates that the robot is in the starting state

Table 17.6 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 17.7 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 17.1, it indicates the robot is in running state

Table 17.8 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 17.9 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 17.10 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”)

18. Process Command

18.1.1 Pallet Commands

Table 18.1 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 18.2 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 18.3 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 18.4 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 18.5 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 18.6 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 18.7 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 18.8 Release pallet command

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

Table 18.9 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 18.10 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 18.1 and Figure 18.2 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.

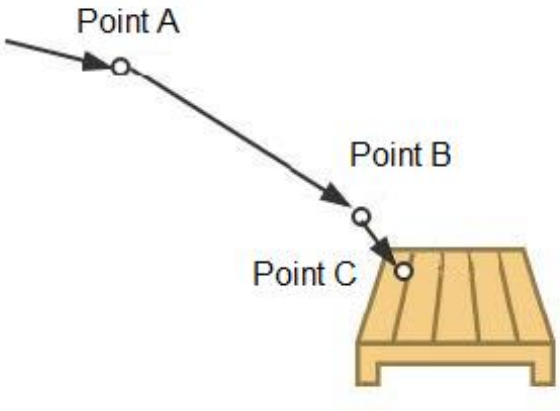

Figure 18.1 Stack assembly path

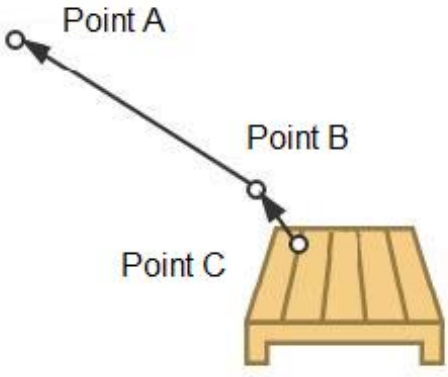

Figure 18.2 Stack dismantling path

18.1.2 Conveyor Tracking Command

Table 18.11 Set conveyor parameter command

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

Table 18.12 Obtain status of the object

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

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

Table 18.13 Set offset command

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

Table 18.14 Set time compensation command

Function	SetCnvTimeCompensation (*time*)
Description	Set time compensation This command is used for compensating the pick-up position offset in the moving direction of the conveyor which is caused by taking photos with a time delay

Parameter	time: time-offset. Unit: ms
Return	0: No error 1: Error

Table 18.15 Synchronize conveyor command

Function	SyncCnv (*CnvID*)
Description	Synchronize the specified conveyor The motion commands used between SyncCnv(*CnvID*) and StopSyncCnv(*CnvID*) only support **Move** command
Parameter	CnvID: Conveyor index
Return	0: No error 1: Error

Table 18.16 Stop synchronizing conveyor command

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

