

1 Overview

1.1 Variables

Variable naming conventions: The variable name is composed of letters, numbers, and underscores and can only start with letters. The name length cannot exceed 32 characters.

Note: The variable name is case sensitive.

The variables are divided as follows according to their scope:

Variable type	Variable data type	Remarks
Global variables	B/R/D/PR/P/PALLET/custom variables	Prefix custom variables with the modifier Global
Module variables	LB/LR/LD/LPR/LP/LPALLET/custom variables	Variables defined inside the module and outside the function body
Local variables	Custom variables	Variables defined inside the function body

Variables are divided as follows according to their numeric type:

Variable type	Variable data type
String type	B/LB/R/LR/D/LD/BOOL/BYTE/INT/FLOAT/DOUBLE
I/O variables	IN/OUT/INB/OUTB/INW/OUTW
String variables	STR/String
Translation variable	PR/LPR
Position variables	P/LP
Pallet variables	Pallet/LPallet

■ Numerical variables

Variable type	Width	Value range	Subscript range
BOOL	1 byte	TRUE, FLASE	Custom variables
BYTE	1 byte	0 to 255	Custom variables
INT	4 bytes	-2147483647 to 2147483647	Custom variables
FLOAT	4 bytes	-2 ¹²⁸ to +2 ¹²⁸	Custom variables
DOUBLE	8 bytes	-2 ¹⁰²⁴ to +2 ¹⁰²⁴	Custom variables
B	Same as BYTE	0 to 255	0 to 255
R	Same as INT	-2147483647 to 2147483647	0 to 255
D	Same as DOUBLE	-2 ¹⁰²⁴ to +2 ¹⁰²⁴	0 to 255
LB	Same as BYTE	0 to 255	0 to 255
LR	Same as INT	-2147483647 to 2147483647	0 to 255
LD	Same as DOUBLE	-2 ¹⁰²⁴ to +2 ¹⁰²⁴	0 to 255

Note: For the numerical variables, the assignment of different types of variables is done through forced conversion. For example:

Byte AAA; AAA = 278; Execution result AAA = 22.

BOOL BBB; BBB = 3; Execution result BBB = TRUE;

Note: The value of the variable is initialized when the variable is defined, and the range of the variable is checked. If the variable exceeds the range during initialization, such as Byte AAA=278, an error will be reported.

For the assignment of the system variables B, R, D, LB, LR, LD, the value is checked against the value range. When the range is exceeded, the execution result will prompt an alarm. When B, R, D, LB, LR or LD is passed into the function as a parameter, a forced conversion will be taken.

■ I/O variables

Variable type	Character unit	Length	Subscript range	Value range	Remarks
IN	bit	1bit	0 to 13823	ON(1)/OFF(0)	0 to 63 for standard I/O; 12800 to 13823 for memory I/O, readable and writeable; and other for read-only I/O
Out	bit	1bit	0 to 13823	ON(1)/OFF(0)	Readable and writeable
InB	Byte	1Byte	0 to 1727	0 to 255	Byte-wise access through the same address as In variable.
OutB	Byte	1Byte	0 to 1727	0 to 255	Byte-wise access through the same address as Out variable.
InW	Word	1Word	0 to 863	0 to 65535	Word-wise access through the same address as In variable.
OutW	Word	1Word	0 to 863	0 to 65535	Word-wise access through the same address as Out variable.

Access to I/O variables

(1) Reading of I/O variables

Bitwise reading

B[0] = In[X]; (X range: 0 to 13823) //1 for ON, 0 for OFF

B[0] = Out[X]; (X range: 0 to 13823) //1 for ON, 0 for OFF

Byte-wise reading:

R[0] = InB[X] (X range: 0 to 1727)

R[0] = OutB[X] (X range: 0 to 1727)

Word-wise reading:

R[0] = InW[X] (X range: 0 to 863)

R[0] = OutW[X] (X range: 0 to 863)

Bitwise reading of byte data:

Out[0] = InB[X].bit[Y] (X range: 0 to 1727, Y range: 0 to 7)

Bitwise reading of word data:

Out[0] = InB[X].bit[Y] (X range: 0 to 863, Y range: 0 to 15)

Reading 4-byte registers or 2-word registers with 32-bit integer:

R[0] = InB[0].Int

R[0] = InW[0].Int

Reading 4-byte registers or 2-word registers with 32-bit FLOAT:

D[0] = InB[0].Float

D[0] = InW[0].Float

Reading 8-byte registers or 4-word registers with 64-bit DOUBLE:

D[0] = InB[0].Double

D[0] = InW[0].Double

Note: INB, INW, OutB and Out variables support access by Int, Float and Double, while bit-type In and Out variables do not.

(2) Writing of I/O variable

Bitwise writing:

Out[X] = ON; (X range: 0 to 13823)

In[X] = OFF; //(When writing, X supports memory I/O in the range of 12800 to 13823)

Note: When the value of the right expression is the value 1, it is equivalent to ON; when the value of the right expression is 0, it is equivalent to OFF, and when it is other values, a running error occurs.

Bytewise writing:

OutB[5] = 3;

InB[X] = 6; //(As the left value, X supports memory I/O in the range of 1600 to 1727)

Wordwise writing:

OutW[5] = 3;

InB[X] = 6; //(As the left value, X supports memory I/O in the range of 800 to 863)

Bitwise writing of byte data:

OutB[X].bit[Y] = ON; (X range: 0 to 1727, Y range: 0 to 7)

InB[X].bit[Y] = ON; (As the left value, X supports memory I/O in the range of 1600 to 1727, and the range of Y is 0 to 7)

Bitwise writing of word data:

OutW[X].bit[Y] = OFF; (X range: 0 to 863, Y range: 0 to 15)

InW[X].bit[Y] = OFF; (As the left value, X supports memory I/O in the range of 800 to 863, and the range of Y is 0 to 15)

Writing 4-byte registers or 2-word registers at once with 32-bit integer:

OutB[X]. Int= <Var> (X range: 0 to 1727)

OutW[X]. Int= <Var> (X range: 0 to 863)

InB[X]. Int = <Var> (As the left value, X supports memory I/O in the range of 1600 to 1727)

InW[X]. Int= <Var> (As the left value, X supports memory I/O in the range of 800 to 863)

Writing 4-byte registers or 2-word registers at once with 32-bit FLOAT:

OutB[X]. Float = <Var> (X range: 0 to 1727)

OutW[X]. Float = <Var> (X range: 0 to 863)

InB[X]. Float = <Var> (As the left value, X supports memory I/O in the range of 1600 to 1727)

InW[X]. Float = <Var> (As the left value, X supports memory I/O in the range of 800 to 863)

Writing 8-byte registers or 4-word registers with 64-bit DOUBLE:

OutB[X]. Double= <Var> (X range: 0 to 1727)

OutW[X]. Double= <Var> (X range: 0 to 863)

InB[X]. Double= <Var> (As the left value, X supports memory I/O in the range of 1600 to 1727)

InW[X]. Double= <Var> (As the left value, X supports memory I/O in the range of 800 to 863)

(3) Logical operation command

Bitwise operators:

& AND; example OUT[0] & OUT[1]

| OR; example OUT[0] | OUT[1]

^ XOR; example OUT[0] ^ OUT[1]

! NOT; example !OUT[1]

<< Left shift OUTB[0] = 1 << 6

>> Right shift OUTB[0] = 8 >> 1

(4) Description of parameters (supporting variables)

Explanation on assigning the In\InB\InW variables:

Only the input of memory I/O supports write operations, that is, only variables within the following range in In\InB\InW support write operations:

In[X] (X range: 12800 to 13823)

InB[X](X range: 1600 to 1727)

InW[X] (X range: 800 to 863)

■ String variables

Variable type	Character unit	Length	Subscript range
STR	Char	256 chars	0 to 255
String	Char	256 chars	Custom variables

■ Position variables

Variable type	Character unit	Length	Subscript range
P	Byte	96	0 to 9999
LP	Byte	96	0 to 9999

■ Translation variable

Variable type	Character unit	Length	Subscript range
PR	Byte	96	0 to 255
LPR	Byte	96	0 to 255

■ Pallet variables

Variable type	Character unit	Length	Subscript range
Pallet	--	--	0 to 255
LPallet	--	--	0 to 255

Example:

Int asd;#Variable definition

Bool sf = true;#Can be initialized when a variable is defined

1.2 Arrays

Supports arrays of various data types, including 2D, 3D arrays, etc.

Format:

BOOL variable name [number of elements]...;

BYTE variable name [number of elements] [number of elements]...;

Number of elements: 1 to 10000. The total number of elements in the array cannot exceed 10000.

Array data initialization:

BOOL variable name [number of elements] = {element1, element2...};

BOOL variable name [number of elements][number of elements] = {{element1, element2...},
{element1, element2...}};

BOOL variable name [number of elements][number of elements][number of elements]=
{{{element1, element2...},{element1, element2...}},{element1, element2...},{element1,
element2...}}};

The initial value for the items missing during initialization is 0.

Example:

Bool Var_Bool = true;

Int Var_Int[2] = {1,3};

Float Var_Float[2][3] = {{1,2,3},{4,5,6}};

Print Var_Bool;

Print Var_Int[0];

Print Var_Int[1];

Print Var_Float[0][0];

Print Var_Float[1][2];

Output:

1 1 3 1 6

1.3 Structs

Format:

```
Struct Struct variable name
...struct element definition
EndStruct
```

Definition of struct variables:

Struct name Struct variable name;

Struct name Struct variable name = {initialization data};

Elements in initialization data are separated by ",", and data nested in arrays or structs are enclosed with {}.

The system has built-in struct variable variables P/LP JP PR/LPR TOOL WOBJ.

The members of the P/LP struct variable are as follows:

P[*].Data[8]

P[*].CrdNo

P[*].ToolNo

P[*].UserNo

P[*].ArmType[4]

PR/LPR/TOOL/USER struct variable members X, Y, Z, A, B, C

Example: PR[*].X Tool[*].X User[*].Y

Example 1:

P[0].Data[0] = 100;

P[0].Data[1] = 200;

P[0].Data[2] = 0;

P[0].Data[3] = 90;

P[0].CrdNo = 2;

P[0].ToolNo = 0;

P[0].UserNo = 1;

P[0].ArmType[0] = 1;

PR[1].X = 134;

Example 2: Use of structs

#The struct definition is written outside the function body at the beginning of the file

Struct AAA

Int var1;#Definition of struct variable member. The value of the variable cannot be initialized at this time. int var1 = 0 is wrong.

Bool var2;

Byte var3[2];

EndStruct;

Start;

#Definition of struct variables

AAA as; #If not initialized, all values in the struct are 0

as.var1 = 6;#Assignment of members in the struct

Print as.var1; #Access to members in the struct

AAA bs = {1,true,{3,4}};#Struct variables can be initialized when defined

End;

1.4 Scope of Variables and Instructions

Variables	Initialization conditions	Initialized value	Scope
Global custom variables	Project compilation	0	Whole project
Global variables B, R, D, STR, PR	Retentive at power off, no initialization	Not restored	Whole system
Global position variable P	Project compilation or synchronization of global point files	Initialized to the value in the project	Whole project
IN, OUT, IG, OG, DA, AD	Re-power	0	Whole system
Local variables LP, LB, LR, LD, LPR, LPallet, String	Project compilation	0/Null	Current program file for the current task
RX buffer data, TX buffer data	Initialized when called with the instruction Open or Close, or when closed or opened through the user interface.	Null	Whole project
Socket or COM connection state	The COM port is closed when the project is compiled, and the Socket needs to be closed through the instruction Close or user interface.	OFF	Whole project

Instruction	Initialization conditions	Initialized value	Scope	Is effective in multitasking?	Is subject to forced blocking?
-------------	---------------------------	-------------------	-------	-------------------------------	--------------------------------

Velset vlaue		Project compilation	/	Between Velset vlaue and Velset OFF or next Velset vlaue in the current program file.	No	No
Velset Rate[value]		Not initialized	/	Between current Velset Rate[value] and next Velset Rate[value]. Note: Velset OFF is not valid for current instruction.	Yes	No
Group IG OG		Power on initialization	/	Whole system	Yes	No
SlewMode		Project compilation	0	Current task	No	No
SetToolParm		Project compilation	Tool parameters set in system	Whole project	No	No
SetUserParm		Project compilation	User parameters set in system	Whole project	No	No
OffSetUserParm		Project compilation	User parameters set in system	Whole project	Yes	No
Motion parameter	SetAccRamp	Project compilation, or return to the start line of the main task, or return to the start line of all taks	Jerk: 50.0 Accelerati on and deceleratio n: 50.0%	Whole project	No	No
Transition parameter	SetFlyMode	Project compilation,	0-FLY_FR EE	Whole project	No	No
	SetFlyPress	or return to the start line of the main task, or	Position transition stress: 100			

		return to the start line of all taks	Pose transition stress: 100			
	SetFlyWait		OFF			
Current limit	AvgCurLmt	Project compilation, or return to the start line of the main task, or return to the start line of all taks	axis: all axes enable:ON ratio: 100	Whole project	No	Yes
	MaxTrqLmt		axis: all axes enable:ON ratio: 100			
Position latch	LatchEnable	Project compilation, or return to the start line of the main task, or return to the start line of all taks	Latch off, latched position cleared	Whole project	No	Yes
	ClearLatchPos					No
Collision detection	SetCollMode	Project compilation, or return to the start line of the main task, or return to the start line of all taks	value: ON mode: 3	Whole project	No	Yes
	SetAxiscoll Mode		axisno:all value:on			
	SetAxisColl Level		axisno:all level:100			
Obj parameter reset	SetGripLoad Mass	Project compilation	Load parameters set in system	Whole project	No	Yes
	SetGripLoad Cog					
	SetGripLoad Orient					
	SetGripLoad Inertia					
	GripLoad					
Tool parameter reset	SetToolMass	Project compilation	Tool parameters set in system	Whole project	No	Yes
	SetToolCog					
	SetToolOrient					
	SetToolInertia					

Coordinate system number settings	SetSysTool No	Globally effective after setup	/	Whole project	No	Yes
	SetSysUser No					
Box parameter reset	WZBoxDef	Project compilation	Initialized to 0 and the interference settings are disabled	Whole project	No	No
	WZSphDef					
	WZCylDef					
	WZLimJDef					
	WZDOSet					
	WZDisable					
	WZEnable					
Optimal trajectory	RapidMove	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	ON for CP and PTP motions for high-speed models by default, and OFF for other models.	Whole project	No	No
	SetAcc		0			
Vibration suppression	SLVSMODE	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	OFF	Whole project	No	No
	SLDataClear	/	/	Whole project	No	No
Robot accuracy mode	SetAccuracy Mode	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	Default mode	Whole project	No	No
LoadPointFromFile GetAPointFromFile		Project compilation	Loaded points cleared	Current program file	Yes	No
CnvVision ON		Project compilation	CnvVision OFF	Whole project	Yes	No

RefSys	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	RefSys Base		No	No
EOffsOn	Current program segment	EOffsOff	Current program	No	No
Pause	/	/	Whole project	Ineffective for Xqt task	Yes
Motion instruction with SLOn/SLOff/SLReset parameters	/	/	Whole project	No	Yes

Note:

1. Projection compilation is triggered when you:

InoRobotLab software programming platform	
1. After modifying the .prj/program file/label/global point file, do one of the following: a. Switch control b. Start debug c. Debug a single step d. Start Play e. Set start line f. Return to start line g. Switch to Play mode Note: Modifying the project file includes i. Modifying any project file on the PC programming platform. ii. Modifying any project file on the teach pendant and immediately switching connection to the PC programming platform before synchronization of the modification to the controller is triggered. 2. In the Sync to Controller dialog: Check any of the label/point file/program.	
Teach pendant	
1. Go to Edit > Config > Save. 2. After modifying the .prj/program file/label/point file, do the following: a. Switch control b. Switch from Edit mode to Play mode c. Switch from Edit mode to Debug mode d. Click Save All. Or click Save on the corresponding project file page. a. Go to Edit > Label or Project > Label, modify the parameters and click Save.	

- b. Go to Project > Program > Command, modify the program file and click Save.
 - c. Go to Project > Program > LP[***], modify the local points and click Save.
 - d. Go to Edit > P[***], modify the global points and click Save.
3. Import global point file
4. Create, delete, rename, paste, import program file successfully (or modify the program file and .prj file);
5. Double click another project in the project list to switch the project;
6. Import a project in the project list and the name of the imported project is the same as the name of the currently active project;
7. Rename the currently active project.

2. What is forced blocking?

For a instruction that is subject to forced blocking, it is triggered only after the motion is completed.

Example:

```
Movj P[0]V[30]Z[0]Tool[0]NWait;
```

```
SetSysToolNo(5);
```

Even if the instruction Movj includes Nwait, the system tool number is switched to 5 only after P[0] is arrived.

1.5 Operators

1.5.1 Arithmetic operators

= Assignment
+ Addition
- Subtraction
* Multiplication
/ Division
% Modulus

1.5.2 Logical operators

&& AND
|| OR
! NOT

1.5.3 Relational operators

== Equality
> Greater than
< Less than
>= Greater than or equal to
<= Less than or equal to
<> Inequality

Note: When determining equality and inequality, considering calculation accuracy, it is recommended not to use double variables; otherwise, it may affect the result of the expression due to calculation accuracy.

1.5.4 Bitwise operators

& AND
| OR
^ XOR
~ NOT
<< Left shift
>> Right shift

1.5.5 Special symbols

Comment
; Semicolon, located at the end of a line, represents the end of a line of statements
: Colon, prompts the following text, used in label (L) instruction, Switch-Case-Default, etc.
, Comma, separates items
" " Double quotes, indicates that the quoted content is a string

2 Instructions

2.1 Keywords

Definition: Keywords are reserved for the system. Keywords cannot be used when defining variable names, file names, and label names. Keywords are not case sensitive.

Keywords are listed in the table below:

A	ABS	ACC	ACOS	AD
AIN	ALARM	ALL	AND	ANGLETORA D
AOUT	ARMCHANG E	ARMTYPE	ASIN	ATAN
AVGCURLMT	B	BASE	BINARY	BIT
BOOL	BOX	BREAK	BTOASCII	BYTE
C	CAPS	CASE	CHECKLOCK	CHECKUNLO CK
CLEAR	CLEARCNVO BJECT	CLEARLATCHP OS	CLOSE	CNVRT
CNVVISION	COM	CONTINUE	CONVEYOR	COPYCNVOB JECT
CORNER	COS	CP	CRDNO	CURRENT
CUSTOMFUN C1	CUSTOMFU NC2	CUSTOMMOVE	CVDATATOPOSE	D
DA	DATA	DEC	DECR	DEFAULT
DELAY	DIST	DO	DOUBLE	DS
DTOSTR	ELSE	ELSEIF	END	ENDFOR
ENDFUNC	ENDIF	ENDPROGRAM INFO	ENDSTRUCT	ENDSWITCH
ENDWHILE	EOFFSOFF	EOFFSON	FALSE	FIN
FINE	FLOAT	FLY_FIX	FLY_FREE	FOR
FUNC	GET	GETACTIVEGR IPLOADNO	GETACTIVETOO LNO	GETALARMN O
GETAPOINTF ROMFILE	GETAT	GETCNVOBJEC T	GETCURPOINT	GETLATCHP OS
GETMODBUS COIL	GETMODBU SREG	GETPLCVARBY TE	GETPLCVARDIN T	GETPLCVARI NT
GETPLCVAR LREAL	GETPORTBU F	GETPORTSTAT E	GETRUNSTATE	GETOCKETN O
GETSTRASCI I	GETTORQUE	GLOBAL	GOTO	GRIPLOAD
GROUP	HALT	HEX	HEXTOSTR	HIGHACCUR ACY

HIGHLEVEL	HOME	IF	IG	IMOV
IMOVJ	IMOV	IN	INB	INCLUDE
INCR	INT	INVERT	INW	J1
J2	J3	J4	J5	J6
J7	J8	JOINT	JUMP	JUMPL
KIND	L	LATCHENABLE	LB	LD
LEFT	LH	LOAD	LOADPOINTFROMFILE	LOADSCREW PARM
LOCKSCREW	LOWCASE	LOWLEVEL	LOWSPEEDHIGH ACCURACY	LP
LPALLET	LPR	LR	MAX	MAXTRQ LIMIT
MH	MID	MIDACCURACY	MIDLEVEL	MIN
MOV	MOVFROMGET	MOVFROMPUT	MOVJ	MOV
MOVTOGET	MOVTOPUT	MSFT	NAME	NORMAL
NOT	NOTES	NWAIT	OBJ	OFF
OFFSET	OFFSETJ	OFFSE	OFFSETUSERPARM	OG
ON	OPEN	OR	OUT	OUTB
OUTW	P	PALLET	PAUSE	PE
PICKV	PORT	POW	PR	PRINT
PROGRAMIN FO	PTOSTR	PTP	PULSE	QUIT
R	RADTOANGLE	RAPIDMOVE	RATE	REFSYS
REPEAT	RESUME	RET	RH	RIGHT
ROBOTNAME	RTOSTR	S	SAVEPOINTFILE	SCREWSTOP
SEND	SET	SETACC	SETACCRAMP	SETACCURACY MODE
SETAXISCOL LLEVEL	SETAXISCOL LMODE	SETCOLLMODE	SETFLYMODE	SETFLYPRES S
SETFLYWAIT	SETGRIPLOAD DCOG	SETGRIPLOADI NERTIA	SETGRIPLOADM ASS	SETGRIPLOAD DORIENT
SETMODBUS COIL	SETMODBUS REG	SETPORTBUF	SETSOCKETREC VTYPE	SETSYSTOOL NO
SETSYSUSER NO	SETTOOLCO G	SETTOOLINER TIA	SETTOOLMASS	SETTOOLORI ENT
SETTOOLPA RM	SETUSERPA RM	SIGNAL	SIN	SLDATA CLEAR
SLEWMODE	SLOFF	SLON	SLRESET	SLVSMODE

SOCKET	SOCKETLIST EN	SPRINTF	SQRT	STARTED
STEP	STR	STRCMP	STRFIND	STRFINDEND
STRGETDAT A	STRING	STRLEN	STRREPLACE	STRREVERS E
STRTOASCII	STRTOD	STRTOHEX	STRTOR	STRUCT
SWITCH	T	TAN	TCP	THEN
TIME	TIMEOUT	TIMESTART	TO	TOOL
TOOLNO	TRIMLEFT	TRIMRIGHT	TRUE	UNLOAD
UNLOCKSCR EW	UNTIL	USER	USERNO	V
VELSET	VERSION	VIN	VOID	VOUT
VXCALIB	WAIT	WAITINPOS	WHILE	WORKBENC H
WORLD	WRITEAPOI NTTOFILE	WZBOXDEF	WZCYLDEF	WZDISABLE
WZDOSET	WZENABLE	WZLIMJDEF	WZSPHDEF	X
XQT	Y	Z	ZR	CRDNO
DATA	TOOLNO	USERNO		

3 Motion Instructions

3.1 Movj

Description: Quickly moves a robot from point to point. The trajectory is usually not in a straight line, and all axes reach the destination simultaneously.

Format: Movj P, V, Z, Tool, [User], [Acc],[Dec], [NWait], [Until In == value], [Out(No,value,Type)...],[SLOn/SLOff/SLReset];

Parameters:

P: Target point to reach.

Common form: P[1],LP[1]

Offset form: OffsetJ(P,PR), OffsetT(P,PR), Offset(P,PR), Offset(P,X,Y...).

In particular, PE can be used in Offset. Example: OffSet (PE,PR), which indicates a shift from the current position of the robot, see the description of PE instruction for details.

Note that if the instruction contains an offset form, there are certain requirements for the subsequent [User] and [Tool] parameters.

See **Offset** for details on the use of the instruction Offset.

Tray point getting form: Pallet(PalletNo, Row, Column, Lay), LPallet(PalletNo, Row, Column, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number, see the instruction **P=Pallet** for details.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed.

Z: Arrival and transition parameter. It describes the transition mode in the process of approaching the target point and leaving the target point.

Available options include Fine, Z[0], Z[1], Z[2], Z[3], Z[4], Z[5], ...Z[200], Z[CP].

Fine: Indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion.

Z[0] to Z[200]: The robot does not stop at the target point, but smooths through the set transition area. Transition length = Transition level * Transition unit length. (To set the transition unit length, go to Set > Motion > RunPara > Corner in InoTeachPad).

Z[CP]: The robot does not stay at the target point and smooths through the set transition area with the maximum transition length.

ZR[m]: Relative transition level, integer, value range 0 to 200, represents the percentage of maximum allowable transition length.

(1) If the current instruction is MovJ, MovL or Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half of the instruction length (namely, motion distance from the current point to the target point), which is consistent with the effect of Z[CP]. When the relative transition level m is greater than 100, the maximum allowable transition length will exceed half of the instruction length. When the ZR parameter is 200, the transition length is the full length of the motion instruction.

(2) If the current instruction is Jump or Jumpl and the additional parameter RH = 0, the maximum allowable transition length will exceed half of the instruction when the relative transition level m is greater than 100. When the ZR parameter is 200, the maximum allowable transition length is the full length of the motion instruction.

(3) If the current instruction is Jump or Jumpl and the additional parameter RH≠0, the maximum allowable transition length is the total length of RH segment. At this time, ZR[100] is the total length of RH segment. When the ZR value is greater than 100, it is consistent with the effect of ZR [100].

Note:

1. When the ZR parameter is 200, the robot completely transitions the current motion to the next motion, and the robot may not pass the original trajectory at all. Because the path of motion in the transition area cannot be guaranteed, make sure that the robot transition path does not interfere with surrounding objects at low speeds.
2. Since the ZR parameter allows for a transition length longer than half the length of the instruction, in rare cases, the transition areas of the start and end points of the instruction may overlap. In this case, the robot will reduce the transition area that is more than half the size of the instruction to ensure that the two transition areas no longer overlap. In particular, if both transition areas at the beginning and end of the instruction are larger than half of the instruction, they will be reduced to half of the instruction at the same time.
3. In particular, since the transition length specified by the ZR parameter is calculated by the percentage of the maximum allowable transition length of the instruction (usually half of the instructed displacement), it is important to note that the theoretical transition length of

the two instructions may be inconsistent when the displacement difference between two adjacent motion commands is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For more information, please see the **Transition Characteristics**.

Tool: Tool coordinate system: Tool [0] to Tool [15].

For more information, see **Extended Use of Tool Parameters**.

User: (Optional parameter) User coordinate system: User[0] to User[15].

For more information, see **Extended Use of User Parameters**.

Acc: (Optional parameter) Specifies acceleration. For example, Acc[50] represents 50% of maximum acceleration.

When this parameter is not used, Acc is the same with V value by default. Minimum effective value is 20%.

Dec: (Optional parameter) Specifies deceleration. For example, Dec[50] represents 50% of maximum deceleration.

When this parameter is not used, Dec is the same with ACC value by default. Minimum effective value is 20%.

NWait: (Optional parameter) Identification of not waiting.

Indicates executing subsequent non-motion instructions immediately before reaching the target point.

This causes non-motion instructions, such as operations, signals, logical judgments, etc., to be executed in advance until the next motion instruction or instruction Delay is encountered.

Description:

1. If the following instruction is a motion instruction, regardless of whether the motion instruction includes NWait, the motion instruction will be issued in advance;
2. When multiple motion instructions are used, regardless of whether the motion instruction includes NWait, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance;

Example of use:

##Set the signal immediately when the motion starts, without waiting for the motion to be completed

Movj P[1],V[30],Z[0],NWait;

Set Out[3],ON;

Note for use with NWait at the end of continuous motion:

For multiple motion instructions, if the last motion instruction includes NWait and is followed by a non-motion instruction, the non-motion instruction will be executed in advance.

Example:

001	▶ START;	001	▶ START;
002	Movj P[0],V[30],Z[0];	002	Movj P[0],V[30],Z[0];
003	Movj P[1],V[30],Z[0];	003	Movj P[1],V[30],Z[0];
004	Movj P[2],V[30],Z[0];	004	Movj P[2],V[30],Z[0];
005	Movj P[3],V[30],Z[0],NWait;	005	Movj P[3],V[30],Z[0],NWait;
006	Set Out[6],ON;	006	Set Out[6],ON;
007	END;	007	Movj P[4],V[30],Z[0];
		008	Movj P[5],V[30],Z[0];
		009	Movj P[6],V[30],Z[0],NWait;
		010	Set Out[7],ON;
		011	END;

On the left, Set Out[6] will be executed before the fifth statement. How soon it is performed in advance is affected by the program preprocessing, and in this case, it will be executed immediately following the first statement.

On the right, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

Until In == value: (Optional parameter) Detects signals in running and stops immediately after conditions are met.

Input signals are detected in the motion process. If conditions are met, motion in the current line is terminated immediately and the subsequent lines continue to be executed. If conditions are not met, motion continues until the end point.

Subparameter:

In: Input signal, only supports In[0]-In[63].

Value: Input signal value, ON or OFF.

Example of use:

##Detect In [5] during motion, and immediately stop the motion when it is ON; Otherwise, keep moving until the end.

Movj P[1],V[30],Z[0], Until In[5] == ON;

Note:

1. If the Until parameter is used, Z is considered Z[0];
2. When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will not be executed in advance;
3. The stop triggered by Until has the same effect as clicking the stop button.
4. When the parameter Until is used, no transition is performed at the start and end points of the current instruction.

Out(No,value,Type)...: (Optional parameter) Parallel output signal during operation.

Up to two signals are output in parallel in a motion instruction.

Subparameter:

No: Output signal serial number, only 0-63 is supported.

Value: Output signal value.

Type: Type of output signals in the motion process. Different motions can use different Types.

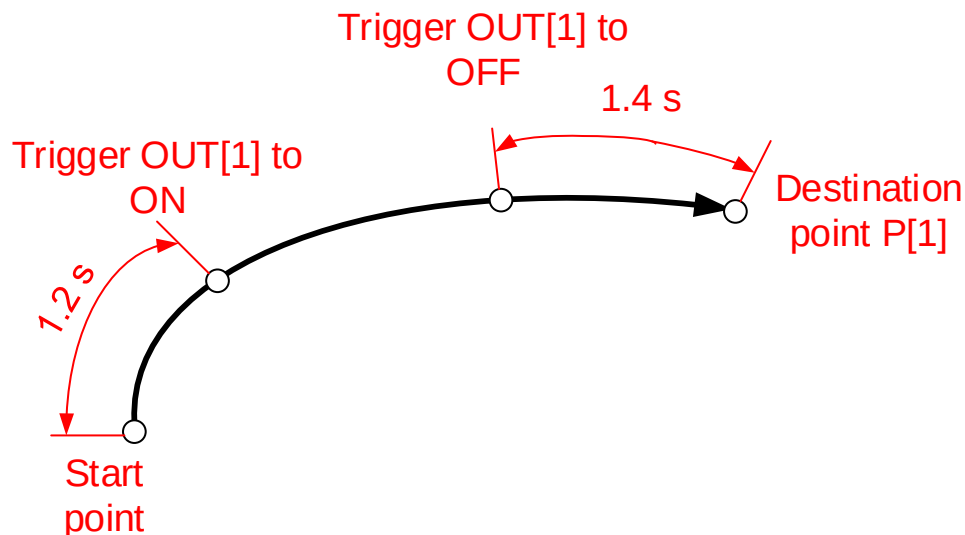
Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] represents the time, unit: seconds, range -65535 to 65535. n >= 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, motion is triggered at the start or end points of this travel section at most, so the set time is invalid.	Movj/ Movl/ Movc/Jump/Jump L
Ds[n]]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when moving to n% of the path. The value ranges from -100.000 to 100.000. The negative number represents the percentage with the endpoint as the reference point.	Movj/ Movl/ Movc/Jump/Jump L
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The range is -65535 to 65535. n >= 0 indicates signals are output after the robot moves to n mm from the start point. n < 0 indicates signals are output before the robot moves to n mm from the end point.	Movl/Movc/Jump L

Note:

1. Precision cannot be guaranteed for motion I/O set in the transition area.
2. When T[n]/Ds[n]/S[n] is set beyond the motion range, motion is triggered at the start and end points of this travel section at most.
3. When Jump/JumpL uses Ds[n] and S[n], only the horizontal movement segment is considered.
4. When using Out (No, value, T[n]), the timer will be reset after pausing.
5. When two motion instructions with the same points are run consecutively in the program, the second motion instruction is invalid.
6. If you want the robot to pass through the singular position, you can only use the instruction Movj.

Example:

Movj P[1] ,V[30], Z[3],OUT(1,ON,T[1.2]), OUT(1,OFF,T[-1.4]);



SLOn/SLOff/SLReset: (optional parameter), a symbol of self-learning of vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;

2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;

2) The time for each self-learning is approximately 1.5s;

3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;

4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;

5) If it is found that the vibration dampening effect does not meet the requirements after using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;

6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;

7) In general, the more obvious the vibration, the better the likelihood of self-learning effect.

However, if the vibration is very obvious, it may lead to self-learning of incorrect data;

8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

1) In the following program, after executing the instruction SLDataClear All, all historical self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

```
SLDataClear All;  
For B[0]=0,B[0]<10,Step[1]  
    Movj LP[0],V[30],Z[0],SLOff;  
    Movj LP[1],V[30],Z[CP],SLOn;  
    Movj LP[2],V[30],Z[CP],NWait,SLReset;  
    Movj LP[3],V[30],Z[0];  
    Set Out[1];  
    SLVSMODE MidLevel;  
EndFor;
```

Description of PE:

PE: current point.

When the trajectory is cleared, the PE is repositioned to indicate the current position of the robot. This is equivalent to taking the current point under the joint coordinate system using the tool number and user number in the system.

When no trajectory clearing operation is performed, the position of PE is the target point of the previous motion instruction. The coordinate system number is based on the joint coordinate system, and the tool number as well as the user number are the same as the previous motion instruction.

Description:

- 1) For motion commands with PE, care should be taken if the previous motion instruction includes Until. When the previous motion instruction includes Until and the motion is not paused or stopped, if the subsequent motion instruction uses Offset (PE, PR), then the point represented by PE is not the current point, but the target point in the previous motion instruction.
- 2) The tool number and user number of the point represented by PE indicate the tool number and user number of the point after the last motion instruction is executed.
For the tool number and user number of the point after the previous motion instruction is executed:
 - a) When the previous motion instruction does not include tool/user parameters, the tool number and user number of the PE are the same with those of the point taken by the previous motion instruction.

Example: The tool number and user number of LP [1] are 1 and 3 respectively, then execute the following instruction:

Movj LP[1], V[30], Z[0];

Movj OffsetT(PE, PR[0]), V[30], Z[0], Tool[1];

The tool number and user number of the PE point are the same as those of LP [1], which are 1 and 3 respectively;

- b) When the previous motion instruction includes parameter Tool/User, the parameters Tool/User of PE are the same with those of the previous motion instruction.

Example: The tool number and user number of LP [1] are 1 and 3 respectively, then execute the following instruction:

Movj LP[1], V[30], Z[0], Tool[5], User[2];

Movj OffsetT(PE, PR[0]), V[30], Z[0], Tool[5];

The tools number and user number of PE point are 5 and 2, respectively.

- 3) When the previous motion instruction of PE is instruction Home or Armchange, the tool number and user number of PE are the same with the tool/user parameters of the instruction Home or Armchange.

Example: Complete the following instructions:

Movj LP[0], V[30], Z[0], Tool[3], User[1];

Movj LP[1], V[30], Z[0];

Home[0];

Movj OffsetT(PE, PR[0]), V[30], Z[0], Tool[5];

The tool number and user number of the PE point are 3 and 1 respectively, regardless of the tool number and user number of LP [1].

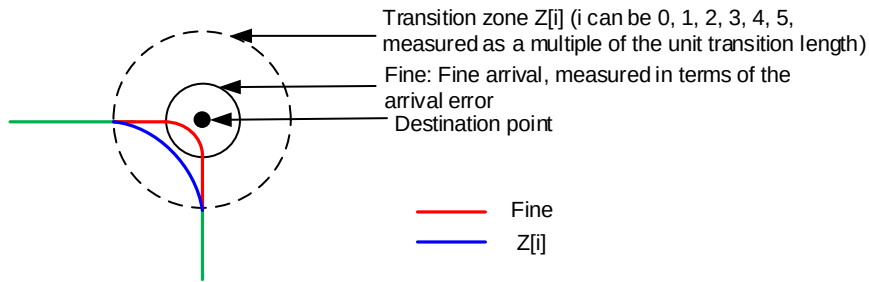
- 4) The instruction OffsetT (PE, **) must include parameter Tool.

Description of transition:

Transition is used to accelerate motion rhythm. The transition range is an area that uses a target point as a center of a circle and a transition radius as a radius.

Transition length = Transition level * Transition unit length.

Type	Method
Fine	It indicates accurate arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion. (See <u>Arrival error</u>)
Z[0]	There is no transition, but the robot cannot be ensured to reach a target point accurately.
Z[1]	There is a transition at one transition unit length.
Z[2]	There is a transition at twice the transition unit length.
Z[3]	There is a transition at three times the transition unit length.
Z[4]	There is a transition at four times the transition unit length.
Z[5]	There is a transition at five times the transition unit length.
Z[CP]	There is a transition at the maximum transition length (Refer to the maximum transition length).



The unit transition length is divided into joint transition unit and linear transition unit. The instructions MoveJ and Jump use joint unit (3-axis Scara robots still use linear unit), while other motion instructions use linear unit.

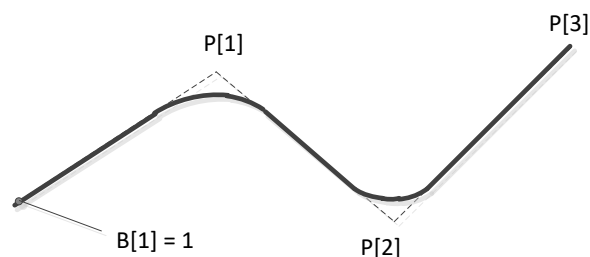
Restrictions on transition: In some cases, using Z[1] to Z[5], Z [CP] to define transitions is invalid and considered as Z[0].

1. For the target point as the end, the actual transition effect is always Z[0].
2. If the motion instruction uses the parameter Until, the actual transition effect is always Z[0].
3. If there are blocking non-motion instructions between motion instructions, the actual effect is always Z [0].
4. For motions that are connected end-to-end in the For and While loops, the actual transition effect is always Z [0]. However, for motions that are connected using L-Goto, the actual transition is still effective.

Note: For motions that are connected end-to-end in the For and While loops, if transition is required, you can use Nwait.

5. There are other instructions between the two motion instructions, such as Set Out, B=1, etc.
 - a) If a motion instruction includes a Nwait parameter, the robot will execute subsequent non-motion instructions before the motion begins. Depending on the execution time of non-motion instructions, if all non-motion instructions between the two motion instructions have already been executed when the robot enters the transition area, the robot will be able to maintain the original transition length.

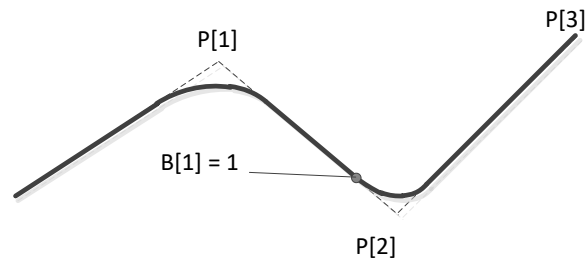
```
Movl P[1],V[30],User[1],Tool[2],Z[3];
Movl P[2],V[30],User[1],Tool[2],Z[3],NWAIT;
B[1] = 1;
Movl P[3],V[30],User[1],Tool[2],Z[0];
```



- b) If the motion instruction does not include a Nwait parameter, in the system default or

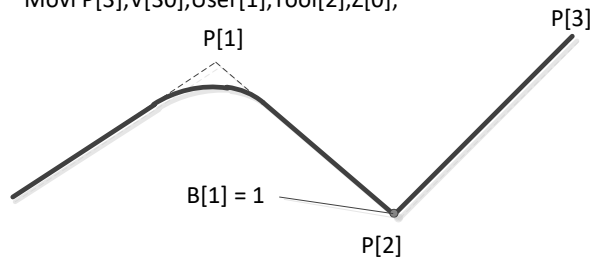
when the transition wait is disabled (SetFlyWait (OFF)), the robot will execute subsequent non-motion instructions before entering the transition area. Depending on the execution time of non-motion instructions, the actual transition length of the robot will likely be shortened or even disappear.

```
Movl P[1],V[30],User[1],Tool[2],Z[3];
Movl P[2],V[30],User[1],Tool[2],Z[3];
B[1] = 1;
Movl P[3],V[30],User[1],Tool[2],Z[0];
```

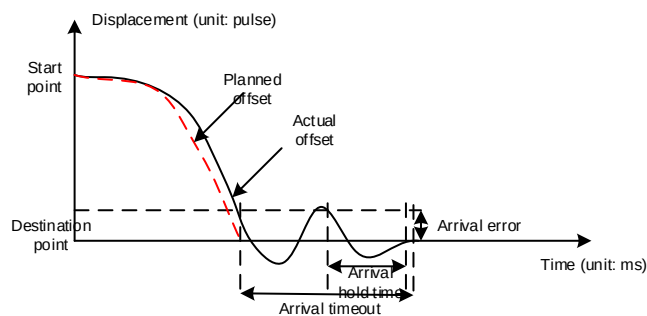


- c) If the motion instruction does not include a Nwait parameter, when transition wait is enabled (SetFlyWait(ON)), the robot will slow down and stop at the target point before executing subsequent non-motion instructions. In this case, the transition disappears.

```
SetFlyWait(ON);
Movl P[1],V[30],User[1],Tool[2],Z[3];
Movl P[2],V[30],User[1],Tool[2],Z[3];
B[1] = 1;
Movl P[3],V[30],User[1],Tool[2],Z[0];
```



Arrival error:

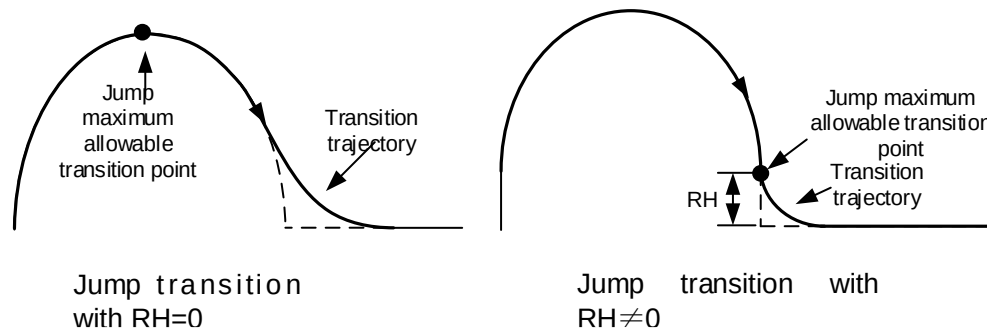


The arrival error includes:

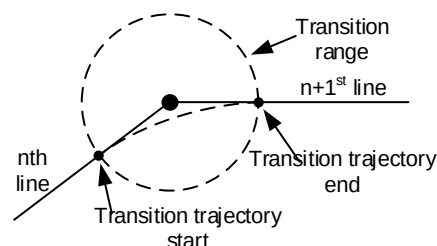
- (1) Arrival error: Default 80,000 pulses, which can be adjusted through settings.
- (2) Arrival hold time: Default 20 pulses, which can be adjusted through settings.
- (3) Arrival timeout: System fixed value 2,000ms.

Maximum transition length: For specific motion, an allowable transition length is limited.

When any set transition length exceeds an allowable maximum transition length, take the maximum transition length as an actual transition length. The maximum allowable transition length of MovL, MoveJ and MovC is a half of the total length. For Jump and JumpL commands, when RH (or LH) is 0, the allowable maximum transition length is a half of the total length, as shown in Figure 1. When RH (or LH) is not 0, the allowable maximum transition length is RH (or LH), as shown in Figure 2.

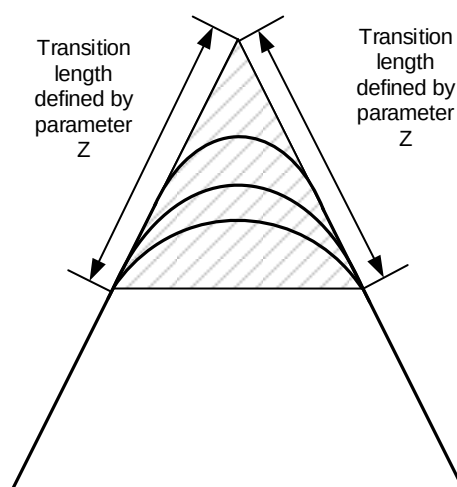


Impact of transition on motion I/O: Effect of transition on motion I/O: When the dotted transition area shown in the following figure exists, the set effective precision of motion I/O in the transition area cannot be guaranteed and the precision actually is effective in an uncertain position between the start point and end point of the transition track.



Considerations when using transition:

As shown in the figure below, the Z parameter defines the transition length, and the transition trajectory is controlled within the shaded triangle in the figure. However, it is not entirely certain and may change with changes in parameters such as speed and acceleration. It is important to confirm the trajectory at the actual operating speed to avoid collisions.

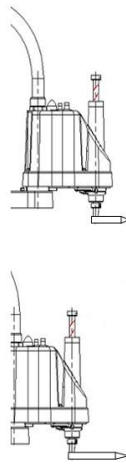


Extended use of Tool parameters:

The Tool parameters in the motion instruction can be used to switch tools. Therefore, when a new tool is used, the new TCP can reach the position and pose of the original TCP.

Application: Commonly used in application scenarios where visual gripping of goods requires calibration of the tool coordinate system parameters by the camera and where the tool coordinate system parameters need to be changed dynamically.

Note: The tool changing function only requires that the tool number of position variables is selected correctly. Position variables can be gotten under any coordinate system, even joint and base coordinate systems.



Case:

Previously, Tool1 was used for teaching to get points and the tool number of the position variable was selected as 1. The programming is as follows:

Movj P[1],V[30],Z[0];		Movj P[1],V[30],Z[0],Tool[1];
Movl P[2],V[30],Z[3];	也可以	Movl P[2],V[30],Z[3],Tool[1];
Movl P[3],V[30],Z[3];		Movl P[3],V[30],Z[3],Tool[1];
.....		

Now the tool is replaced and Tool2 is used for processing. You need not perform teaching again and can directly change Tool in the instruction:

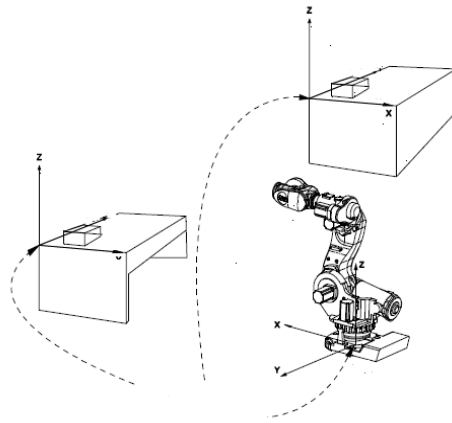
```
MovjP[1],V[30],Z[0],Tool2;  
MovlP[2],V[30],Z[3],Tool[2];  
MovlP[3],V[30],Z[3],Tool[2];
```

Extended use of User parameters:

You can switch the user coordinate system by the User parameter in the motion instruction.

Application: Applied to application scenarios where there are identical working points with determined relative positions on different work tables.

Note: The important prerequisite is that the points in the original program are obtained under the user coordinate system.



Case:

For an original batch of work objects below the table, establish User1 on the table and use User1 as the current user coordinate system. Then perform teaching, point getting and programming under the user coordinate system.

Movj P[1],V[30],Z[0];		Movj P[1],V[30],Z[0],User[1];
Movl P[2],V[30],Z[3];	也可以	Movl P[2],V[30],Z[3],User[1];
Movl P[3],V[30],Z[3];		Movl P[3],V[30],Z[3],User[1];
.....		

Later, the position of the work table was changed and User2 was established on the new work table. You can directly change as follows:

```
MovjP[1],V[30],Z[0],User[2];
MovlP[2],V[30],Z[3],User[2];
MovlP[3],V[30],Z[3],User[2];
```

Note: When the coordinate system numbers of P are 1, 2, and 3, the User cannot be switched.

Specifications for duplicate points:

- (1) If the difference in joint angle between two adjacent points is less than 1e-5 radians, they are considered duplicate points.
- (2) When considered as duplicate points, the second motion instruction in the program is invalid, including the motion parameters such as motion I/O, etc.

3.2 Movl

Function: Linear interpolation

Description: Move linearly to a given target position.

Format: Movl P, V, Z, Tool, [User], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)...], [SLOn/SLOff/SLReset];

Parameters:

P: Target point to reach.

Common form: P[1], LP[1], etc. It can also be in offset form and pallet point getting form.

Offset form - OffsetJ(P,PR), OffsetT(P,PR), Offset(P,PR), Offset(P,X,Y...)

In particular, PE can be used in Offset to indicate offset from the current position.

Note that if the instruction contains an offset form, there are certain requirements for the subsequent [User] and [Tool] parameters.

For details, see **Offset**.

Pallet point getting form - Pallet(PalletNo, Row, Column, Lay), LPallet(PalletNo, Row, Column, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number, see the instruction **P=PalletP=Pallet** for details.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed.

Z: Arrival and transition parameter. It describes the transition mode in the process of approaching the target point and leaving the target point.

Fine: Indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion.

Z[0] to Z[200]: The robot does not stop at the target point, but smooths through the set transition area. Transition length = Transition level * Transition unit length. (To set the transition unit length, go to Set > Motion > RunPara > Corner)

Z[CP]: The robot does not stay at the target point and smooths through the set transition area with the maximum transition length.

ZR[m]: Relative transition level, integer, value range 0 to 200, represents the percentage of maximum allowable transition length.

(1) If the current instruction is MovJ, MovL or Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half of the instruction length, which is consistent with the effect of Z[CP]. When the relative transition level m is greater than 100, the transition length will exceed half of the instruction. When the ZR parameter is 200, the transition length is the full length of the motion instruction.

(2) If the current instruction is Jump or Jumpl and the additional parameter RH = 0, the transition length will exceed half of the instruction when the relative transition level m is greater than 100. When the ZR parameter is 200, the maximum allowable transition length is the full length of the motion instruction.

(3) If the current instruction is Jump or Jumpl and the additional parameter RH≠0, the maximum allowable transition length is the total length of RH segment. At this time, ZR[100] is the total length of RH segment. When the ZR value is greater than 100, it is consistent with the effect of ZR [100].

Note:

1. When the ZR parameter is 200, the robot completely transitions the current motion to the next motion, and the robot may not pass the original trajectory at all. Because the path of motion in the transition area cannot be guaranteed, make sure that the robot transition path does not interfere with surrounding objects at low speeds.

2. Since the ZR parameter allows for a transition length longer than half the length of the instruction, in rare cases, the transition areas of the start and end points of the instruction may overlap. In this case, the robot will reduce the transition area that is more than half the size of the instruction to ensure that the two transition areas no longer overlap. In particular, if both transition areas at the beginning and end of the instruction are larger than half of the instruction, they will be reduced to half of the instruction at the same time.
3. In particular, since the transition length specified by the ZR parameter is calculated by the percentage of the maximum allowable transition length of the instruction (usually half of the instructed displacement), it is important to note that the theoretical transition length of the two instructions may be inconsistent when the displacement difference between two adjacent motion commands is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see **Transition feature**.

Tool: Tool coordinate system: Tool [0] to Tool [15].

For details, see **Extended use of Tool parameters**.

User: (Optional parameter) User coordinate system: User[0] to User[15].

For details, see **Extended use of User parameters**.

Acc: (Optional parameter) Specifies acceleration. For example, Acc[50] represents 50% of maximum acceleration.

When this parameter is not used, Acc is the same with V value by default. Minimum effective value is 20%.

Dec: (Optional parameter) Specifies deceleration. For example, Dec[50] represents 50% of maximum deceleration.

When this parameter is not used, Dec is the same with ACC value by default. Minimum effective value is 20%.

NWait: (Optional parameter) Identification of not waiting.

This item indicates that subsequent non-motion commands are immediately executed without running to a target point.

This causes subsequent non-motion commands such as operation, signal and logic judgment to be executed in advance until the next motion command or Delay command is encountered.

Description:

1. If the following instruction is a motion instruction, regardless of whether the motion instruction includes NWait, the motion instruction will be issued in advance;
2. When multiple motion instructions are used, regardless of whether the motion instruction includes NWait, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance;
3. The stop triggered by Until has the same effect as clicking the stop button.
4. When the parameter Until is used, no transition is performed at the start and end points of the current instruction.

Example of use:

##Set the signal immediately when the motion starts without waiting for the motion to be completed

```
Movl P[1],V[30],Z[0],NWait;
Set Out[3],ON;
```

Note for use with NWait at the end of continuous motion:

For multiple motion instructions, if the last motion instruction includes NWait followed by a non-motion instruction, the non-motion instruction will be executed in advance. Example:

001	▶ START;	001	▶ START;
002	Movj P[0],V[30],Z[0];	002	Movj P[0],V[30],Z[0];
003	Movj P[1],V[30],Z[0];	003	Movj P[1],V[30],Z[0];
004	Movj P[2],V[30],Z[0];	004	Movj P[2],V[30],Z[0];
005	Movj P[3],V[30],Z[0],NWait;	005	Movj P[3],V[30],Z[0],NWait;
006	Set Out[6],ON;	006	Set Out[6],ON;
007	END;	007	Movj P[4],V[30],Z[0];
		008	Movj P[5],V[30],Z[0];
		009	Movj P[6],V[30],Z[0],NWait;
		010	Set Out[7],ON;
		011	END;

On the left, Set Out[6] will be executed before the fifth statement. How soon it is performed in advance is affected by the program preprocessing, and in this case, it will be executed immediately following the first statement.

On the right, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

Until In == value: (Optional parameter) Detects signals in running and stops immediately after conditions are met.

Input signals are detected in the motion process. If conditions are met, motion in the current line is terminated immediately and the subsequent lines continue to be executed. If conditions are not met, motion continues until the end point.

Subparameter:

In: Input signal, only In[0]-In[63] is supported.

Value: Input signal value, ON or OFF.

Example of use:

##Detect In [5] during motion, and immediately stop the motion when it is ON; Otherwise, keep moving until the end.

```
Movl P[1],V[30],Z[0], Until In[5] == ON;
```

Note:

1. If the Until parameter is used, Z is considered Z[0];
2. When an instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will not be executed in advance.

Out(No,value,Type)...: (Optional parameter) Parallel output signal during operation.

Up to two signals are output in parallel in a motion instruction.

Subparameter:

No: Output signal serial number, only 0-63 is supported.

Value: Output signal value.

Type: Type of output signals in the motion process. Different motions can use different Types.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] represents the time, unit: seconds, range -65535 to 65535. n >= 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, motion is triggered at the start or end points of this travel section at most, so the set time is invalid.	Movj/ Movl/ Movc/Jump/Jump L
Ds[n]]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when moving to n% of the path. The range is 0.000 to 100.000.	Movj/ Movl/ Movc/Jump/Jump L
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The range is -65535 to 65535. n >= 0 indicates signals are output after the robot moves to n mm from the start point. n < 0 indicates signals are output before the robot moves to n mm from the end point.	Movl/Movc/Jump L

Note:

1. Precision cannot be guaranteed for motion I/O set in the transition area.
2. When T[n]/D[n]/S[n] is set beyond the motion range, motion is triggered at the start and end points of this travel section at most.
3. When Jump/JumpL uses D[n] and S[n], only the horizontal movement segment is considered.
4. When using Out (No, value, T[n]), the timer will be reset after pausing.
5. When two motion instructions with the same points are run consecutively in the program, the second motion instruction is invalid.

Example of use:

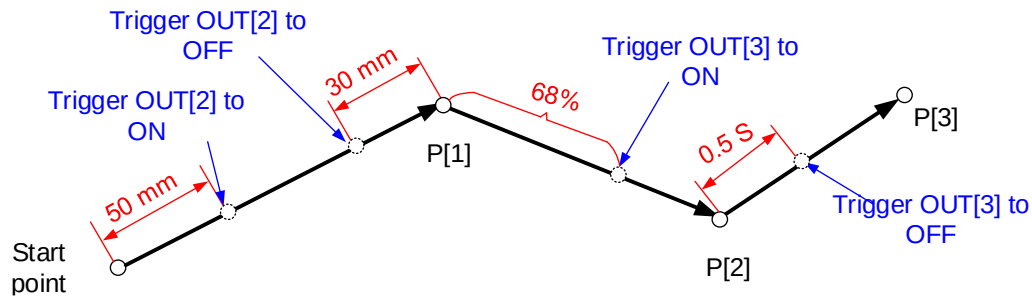
START;

Movl P[1] ,V[30], Z[0],OUT(2,ON,S[50]), OUT(2,OFF,S[-30]);

Movl P[2] ,V[30], Z[0],OUT(3,ON,Ds[68]);

Movl P[3] ,V[30], Z[0],OUT(3,OFF,T[0.5]);

END;



SLOn/SLOff/SLReset: (Optional parameter), a symbol of self-learning for vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

- 1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;
- 2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

- 1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;
- 2) The time for each self-learning is approximately 1.5s;
- 3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;
- 4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;
- 5) If it is found that the vibration dampening effect does not meet the requirements after using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;
- 6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;
- 7) In general, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is very obvious, it may lead to self-learning of incorrect data;
- 8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

- 1) In the following program, after executing the instruction SLDataClear All, all historical

self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

```
SLDataClear All;
  For B[0]=0,B[0]<10,Step[1]
    Movj LP[0],V[30],Z[0],SLOff;
    Movj LP[1],V[30],Z[CP],SLOn;
    Movj LP[2],V[30],Z[CP],NWait,SLReset;
    Movj LP[3],V[30],Z[0];
    Set Out[1];
    SLVSMODE MidLevel;
  EndFor;
```

3.3 Move

Function: Circular interpolation

Description: Moves to a given target position by circular motion

Format: Move P, V, Z, Tool,[User], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)...], [SLOn/SLOff/SLReset];

Parameters:

P: Target point to reach.

Common form: P[1], LP[1], etc. It can also be in offset form and pallet point getting form.

Offset form - OffsetJ(P,PR), OffsetT(P,PR), Offset(P, PR), Offset(P,X,Y...)

In particular, PE can be used in Offset to indicate offset from the current position.

Note that if the instruction contains an offset form, there are certain requirements for the subsequent [User] and [Tool] parameters.

For details, see **Offset**.

Pallet point getting form - Pallet(PalletNo, Row, Column, Lay), LPallet(PalletNo, Row, Column, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number, see the instruction P=Pallet for details.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed.

Z: Arrival and transition parameter. It describes the transition mode in the process of approaching the target point and leaving the target point.

Fine: Indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion.

Z[0] to Z[200]: The robot does not stay at the target point, but smooths through the set transition area. Transition length = transition level * transition unit length. (To set the transition unit length, go to Set > Motion > RunPara > Corner)

Z[CP]: The robot does not stay at the target point and smooths through the set transition area with the maximum transition length.

ZR[m]: Relative transition level, integer, value range 0 to 200, represents the percentage of maximum allowable transition length.

- (1) If the current instruction is MovJ, MovL or Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half of the instruction length, which is consistent with the effect of Z[CP]. When the relative transition level m is greater than 100, the maximum allowable transition length will exceed half of the instruction. When the ZR parameter is 200, the transition length is the full length of the motion instruction.
- (2) If the current instruction is a Jump 'Jump' instruction and the additional parameter RH = 0, the maximum allowable transition length will exceed half of the instruction when the relative transition level m is greater than 100. When the ZR parameter is 200, the maximum allowable transition length is the full length of the motion instruction.
- (3) If the current instruction is Jump or JumpI and the additional parameter RH≠0, the maximum allowable transition length is the total length of RH segment. At this time, ZR[100] is the total length of RH segment. When the ZR value is greater than 100, it is consistent with the effect of ZR [100].

Note:

1. When the ZR parameter is 200, the robot completely transitions the current motion to the next motion, and the robot may not pass the original trajectory at all. Because the path of motion in the transition area cannot be guaranteed, make sure that the robot transition path does not interfere with surrounding objects at low speeds.
2. Since the ZR parameter allows for a transition length longer than half the length of the instruction, in rare cases, the transition areas of the start and end points of the instruction may overlap. In this case, the robot will reduce the transition area that is more than half the size of the instruction to ensure that the two transition areas no longer overlap. In particular, if both transition areas at the beginning and end of the instruction are larger than half of the instruction, they will be reduced to half of the instruction at the same time.
3. In particular, since the transition length specified by the ZR parameter is calculated by the percentage of the maximum allowable transition length of the instruction (usually half of the instructed displacement), it is important to note that the theoretical transition length of the two instructions may be inconsistent when the displacement difference between two adjacent motion commands is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see **Transition feature**.

Tool: Tool coordinate system: Tool [0] to Tool [15].

For details, see **Extended use of Tool parameters**.

User: (Optional parameter) User coordinate system: User[0] to User[15].

For details, see **Extended use of User parameters**.

Acc: (Optional parameter) Specifies acceleration. For example, Acc[50] represents 50% of maximum acceleration.

When this parameter is not used, Acc is the same with V value by default. Minimum effective

value is 20%.

Dec: (Optional parameter) Specifies deceleration. For example, Dec[50] represents 50% of maximum deceleration.

When this parameter is not used, Dec is the same with ACC value by default. Minimum effective value is 20%.

NWait: (Optional parameter) Identification of not waiting.

This item indicates that subsequent non-motion commands are immediately executed without running to a target point.

This causes subsequent non-motion commands such as operation, signal and logic judgment to be executed in advance until the next motion command or Delay command is encountered.

Description:

1. If the following instruction is a motion instruction, regardless of whether the motion instruction includes NWait, the motion instruction will be issued in advance;
2. When multiple motion instructions are used, regardless of whether the motion instruction includes NWait, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance;

Note for use with NWait at the end of continuous motion:

For multiple motion instructions, if the last motion instruction includes NWait followed by a non-motion instruction, the non-motion instruction will be executed in advance. Example:

001	▶ START;	001	▶ START;
002	Movj P[0],V[30],Z[0];	002	Movj P[0],V[30],Z[0];
003	Movj P[1],V[30],Z[0];	003	Movj P[1],V[30],Z[0];
004	Movj P[2],V[30],Z[0];	004	Movj P[2],V[30],Z[0];
005	Movj P[3],V[30],Z[0],NWait;	005	Movj P[3],V[30],Z[0],NWait;
006	Set Out[6],ON;	006	Set Out[6],ON;
007	END;	007	Movj P[4],V[30],Z[0];
		008	Movj P[5],V[30],Z[0];
		009	Movj P[6],V[30],Z[0],NWait;
		010	Set Out[7],ON;
		011	END;

On the left, Set Out[6] will be executed before the fifth statement. How soon it is performed in advance is affected by the program preprocessing, and in this case, it will be executed immediately following the first statement.

On the right, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

Until In == value: (Optional parameter) Detects signals in running and stops immediately after conditions are met.

Input signals are detected in the motion process. If conditions are met, motion in the current line is terminated immediately and the subsequent lines continue to be executed. If conditions are not met, motion continues until the end point.

Subparameter:

In: Input signal, only supports In[0]-In[63].

Value: Input signal value, ON or OFF.

Example of use:

##Detect In [5] during motion, and immediately stop the motion when it is ON; Otherwise, keep moving until the end.

Movc P[1],V[30],Z[0], Until In[5] == ON;

Movc P[2],V[30],Z[0], Until In[5] == ON;

Note:

1. If the Until parameter is used, Z is considered Z[0];
2. When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will not be executed in advance;
3. The stop triggered by Until has the same effect as clicking the stop button.
4. When the Until parameter is used, no transition is performed at the start and end points of the current instruction.

Out(No,value,Type)...: (Optional parameter) Parallel output signal during operation.

Up to two signals are output in parallel in a motion instruction.

Subparameter:

No: Output signal serial number, only 0-63 is supported.

Value: Output signal value.

Type: Type of output signals in the motion process. Different motions can use different Types.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] represents the time, unit: seconds, range -65535 to 65535. n >= 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, motion is triggered at the start or end points of this travel section at most, so the set time is invalid.	Movj/ Movl/ Movc/Jump/Jump L
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when moving to n% of the path. The range is 0.000 to 100.000.	Movj/ Movl/ Movc/Jump/Jump L
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The range is -65535 to 65535. n >= 0 indicates signals are output after the robot moves to n mm from the start point. n < 0 indicates signals are output before the robot moves to n mm from the end point.	Movl/Movc/Jump L

Note:

1. Precision cannot be guaranteed for motion I/O set in the transition area.
2. When T[n]/D[n]/S[n] is set beyond the motion range, motion is triggered at the start and end points of this travel section at most.
3. When Jump/JumpL uses D[n] and S[n], only the horizontal movement segment is considered.
4. When using Out (No, value, T[n]), the timer will be reset after pausing.
5. When two motion instructions with the same points are run consecutively in the program, the second motion instruction is invalid.

SLOn/SLOff/SLReset: (Optional parameter), a symbol of self-learning for vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;

2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;

2) The time for each self-learning is approximately 1.5s;

3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;

4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;

5) If it is found that the vibration dampening effect does not meet the requirements after using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;

6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;

7) In general, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is very obvious, it may lead to self-learning of incorrect data;

8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter

and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

1) In the following program, after executing the instruction SLDataClear All, all historical self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

```
SLDataClear All;
For B[0]=0,B[0]<10,Step[1]
  Movj LP[0],V[30],Z[0],SLOff;
  Movj LP[1],V[30],Z[CP],SLOn;
  Movj LP[2],V[30],Z[CP],NWait,SLReset;
  Movj LP[3],V[30],Z[0];
  Set Out[1];
  SLVSMODE MidLevel;
EndFor;
```

Note:

1. Three points determine an arc, so the instruction Movc must appear in pairs and cannot be separated by any instructions. The previous instruction before Movc must be a motion instruction (Movj, Movl, Move, Jump, JumpL), otherwise an error message "Lack of motion instruction before arc motion" will be reported. These requirements must be followed. Even a comment can affect the results and cause alarms.
2. For two instructions Movc in a segment of circular motion, all parameters except P are subject to the second one.
3. Do not use Movc in combination with PE as it may cause exceptions. Typical example: Movc offset(PE, PR0)...
4. For instructions Movc that appear for the first time in a program, the previous motion instruction must not be used in combination with PE, for example:

Movc instruction that first
appeared in the program:

```
Movj P[1],V[30],Z[0];
Movc P[2],V[30],Z[0];
Movc P[3],V[30],Z[0];
```



```
Movj Offset(PE, PR0),V[30],Z[0];
Movc P[2],V[30],Z[0];
Movc P[3],V[30],Z[0];
```



5. When not used in conjunction with PE, it can be used for Play and single-step Teach. Even if you press pause halfway, the robot can continue to move to the endpoint of the arc.

Example:

START;

```

Movj    P[1] ,V[80] ,Z[0];          ##Moves quickly to P[1]
Movc    P[2] ,V[80] ,Z[3];          ##Moves through P[2] to P[3] and the trajectory is
circular
Movc    P[3] ,V[80] ,Z[3];
END;

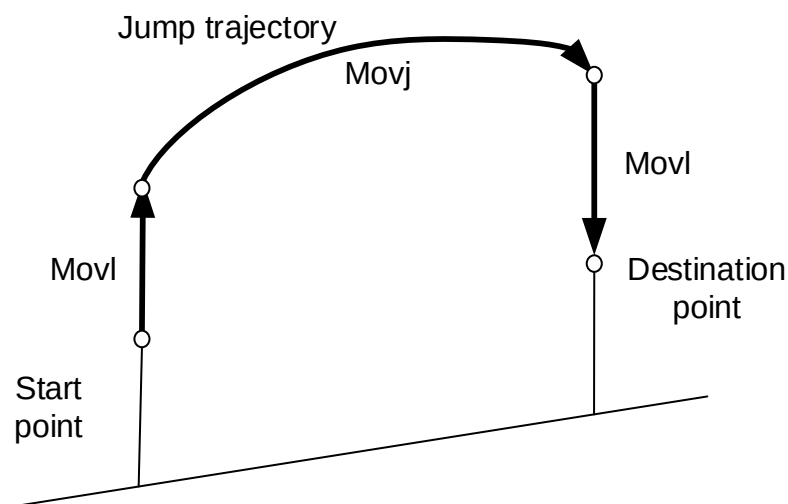
```

6. The first Movc only provides the data of the transition point, and the overall motion effect is only based on the parameters in the second movec.

3.4 Jump

Function: Jump instruction

Description: The motion process is divided into three segments: the starting ascending segment Movl, the middle segment Movj and the descending segment Movl.



Format: Jump P, V, Z, Tool, [User], , [Acc], [Dec], [NWait], [Out(No,value,Type)...], LH, MH, RH,[SLOn/SLOff/SLReset];

Parameters:

P: Target point to reach.

Common form: P[1], LP[1], etc. It can also be in offset form and pallet point getting form.

Offset form -OffsetJ(P , PR), OffsetT(P , PR), Offset(P , PR), OffSet(P,X,Y,..) and so on.

In particular, PE can be used in Offset to indicate offset from the current position.

Note that if the instruction contains an offset form, there are certain requirements for the subsequent [User] and [Tool] parameters.

For details, see **Offset**.

Pallet point getting form - Pallet(PalletNo, Row, Column, Lay), LPallet(PalletNo, Row, Column, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number, see the instruction **P=Pallet** P=Palletfor details.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed.

Z: Arrival and transition parameter. It describes the transition mode in the process of approaching the target point and leaving the target point.

Fine: Indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion.

Z[0] to Z[200]: The robot does not stay at the target point, but smooths through the set transition area. Transition length = transition level * transition unit length. (To set the transition unit length, go to Set > Motion > RunPara > Corner)

Z[CP]: The robot does not stay at the target point and smooths through the set transition area with the maximum transition length.

ZR[m]: Relative transition level, integer, value range 0 to 200, represents the percentage of maximum allowable transition length.

- (1) If the current instruction is MovJ, MovL or Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half of the instruction length, which is consistent with the effect of Z[CP]. When the relative transition level m is greater than 100, the maximum allowable transition length will exceed half of the instruction. When the ZR parameter is 200, the transition length is the full length of the motion instruction.
- (2) If the current instruction is Jump or JumpL and the additional parameter RH = 0, the maximum allowable transition length will exceed half of the instruction when the relative transition level m is greater than 100. When the ZR parameter is 200, the maximum allowable transition length is the full length of the motion instruction.
- (3) If the current instruction is Jump or JumpL and the additional parameter RH≠0, the maximum allowable transition length is the total length of RH segment. At this time, ZR[100] is the total length of RH segment. When the ZR value is greater than 100, it is consistent with the effect of ZR [100].

Note:

1. When the ZR parameter is 200, the robot completely transitions the current motion to the next motion, and the robot may not pass the original trajectory at all. Because the path of motion in the transition area cannot be guaranteed, make sure that the robot transition path does not interfere with surrounding objects at low speeds.
2. Since the ZR parameter allows for a transition length longer than half the length of the instruction, in rare cases, the transition areas of the start and end points of the instruction may overlap. In this case, the robot will reduce the transition area that is more than half the size of the instruction to ensure that the two transition areas no longer overlap. In particular, if both transition areas at the beginning and end of the instruction are larger than half of the instruction, they will be reduced to half of the instruction at the same time.
3. In particular, since the transition length specified by the ZR parameter is calculated by the percentage of the maximum allowable transition length of the instruction (usually half of the instructed displacement), it is important to note that the theoretical transition length of the two instructions may be inconsistent when the displacement difference between two adjacent motion commands is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see **Transition feature**.

Tool: Tool coordinate system: Tool [0] to Tool [15].

For details, see **Extended use of Tool parameters**.

User: (Optional parameter) User coordinate system: User[0] to User[15].

For details, see **Extended use of User parameters**.

Acc: (Optional parameter) Specifies acceleration. For example, Acc[50] represents 50% of maximum acceleration.

When this parameter is not used, Acc is the same with V value by default. Minimum effective value is 20%.

Dec: (Optional parameter) Specifies deceleration. For example, Dec[50] represents 50% of maximum deceleration.

When this parameter is not used, Dec is the same with ACC value by default. Minimum effective value is 20%.

NWait: (Optional parameter) Identification of not waiting.

This item indicates that subsequent non-motion commands are immediately executed without running to a target point.

This causes subsequent non-motion commands such as operation, signal and logic judgment to be executed in advance until the next motion command or Delay command is encountered.

Description:

1. If the following instruction is a motion instruction, regardless of whether the motion instruction includes NWait, the motion instruction will be issued in advance;
2. When multiple motion instructions are used, regardless of whether the motion instruction includes NWait, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance;

Example of use:

##Set the signal immediately when the motion starts without waiting for the motion to be completed

Jump P[1],V[30],Z[0],NWait,LH[60],MH[130],RH[62];

Set Out[3],ON;

Note for use with NWait at the end of continuous motion:

For multiple motion instructions, if the last motion instruction includes NWait followed by a non-motion instruction, the non-motion instruction will be executed in advance. Example:

001	▶ START;	001	▶ START;
002	Movj P[0],V[30],Z[0];	002	Movj P[0],V[30],Z[0];
003	Movj P[1],V[30],Z[0];	003	Movj P[1],V[30],Z[0];
004	Movj P[2],V[30],Z[0];	004	Movj P[2],V[30],Z[0];
005	Movj P[3],V[30],Z[0],NWait;	005	Movj P[3],V[30],Z[0],NWait;
006	Set Out[6],ON;	006	Set Out[6],ON;
007	END;	007	Movj P[4],V[30],Z[0];
		008	Movj P[5],V[30],Z[0];
		009	Movj P[6],V[30],Z[0],NWait;
		010	Set Out[7],ON;
		011	END;

On the left, Set Out[6] will be executed before the fifth statement. How soon it is performed in advance is affected by the program preprocessing, and in this case, it will be executed immediately following the first statement.

On the right, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

Out(No,value,Type)...: (Optional parameter) Parallel output signal during operation.

Up to two signals are output in parallel in a motion instruction.

Subparameter:

No: Output signal serial number, only 0-63 is supported.

Value: Output signal value.

Type: Type of output signals in the motion process. Different motions can use different Types.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] represents the time, unit: seconds, range -65535 to 65535. n >= 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, motion is triggered at the start or end points of this travel section at most, so the set time is invalid.	Movj/ Movl/ Movc/Jump/Jump L
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when moving to n% of the path. The range is 0.000 to 100.000.	Movj/ Movl/ Movc/Jump/Jump L
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The range is -65535 to 65535. n >= 0 indicates signals are output after the robot moves to n mm from the start point. n < 0 indicates signals are output before the robot moves to n mm from the end point.	Movl/Movc/Jump L

Note:

1. Precision cannot be guaranteed for motion I/O set in the transition area.
2. When T[n]/D[n]/S[n] is set beyond the motion range, motion is triggered at the start and end points of this travel section at most.
3. When Jump/JumpL uses D[n] and S[n], only the horizontal movement segment is considered.
4. When using Out (No, value, T[n]), the timer will be reset after pausing.
5. When two motion instructions with the same points are run consecutively in the program,

the second motion instruction is invalid.

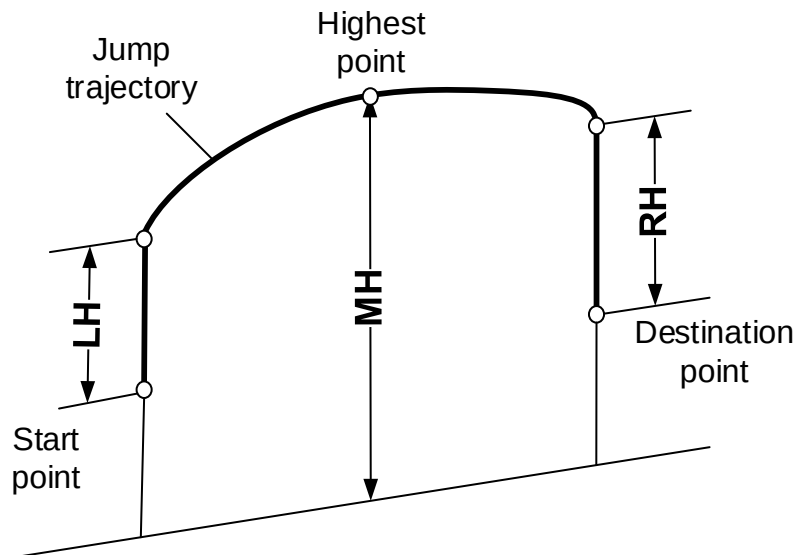
See the following figure.

LH: Vertical motion height at the starting point position; format: LH[Z] , value range 0.000 to 2000.000

RH: Vertical motion height at the destination point position; format: RH[Z] , value range 0.000 to 2000.000

MH: Total height of motion. Format:

a) MH[Z] , value range -2000.000 to 2000.000



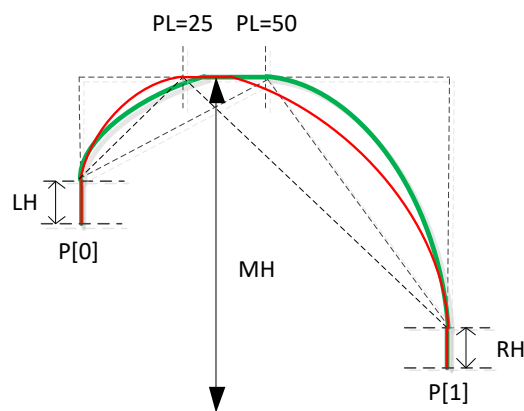
b) MH[Z] [PL]: Sets the Z-axis height limit and the proportion PL of the highest point position for the instruction Jump.

PL: The proportion of the highest point position to the horizontal displacement in the path of Jump motion.

(1) Parameter type: Integer

(2) Value range: 20 to 80

(3) Default value: 50 (horizontal midpoint)



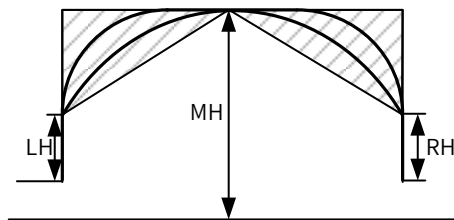
Note:

1. The PL parameter is only valid for the current Jump motion. If you need to adjust the

highest point for multiple Jump commands, please set the additional parameters for the corresponding instruction.

2. It is recommended to use the default value (50) when requirements on efficiency or compliance are not high. When you need to adjust the PL parameter, please check whether the robot path interferes with the surrounding objects at low speed first. For the initial setup, please follow the default value of 50 for trial run and gradually adjust according to the Z-axis displacement ratio on both sides of Jump to avoid the robot path interfering with the surrounding environment.
3. When the proportion of the highest point is set too large or too small, the robot's motion efficiency may be reduced, so be careful to adjust the highest point position reasonably.
4. In particular, when the horizontal distance between the starting point and the ending point of the instruction Jump is large, the robot may move horizontally at the maximum height specified by the MH parameter. In this case, the highest point corresponding to the PL parameter will be included in the horizontal displacement motion path.

In the range beyond RH/LH, motion in both vertical and horizontal directions will occur, and the trajectory formed is controlled within the shaded triangle. However, it is not entirely certain and may change with changes in parameters such as speed and acceleration. It is important to confirm the trajectory at the actual operating speed to avoid collisions.



SLOn/SLOff/SLReset: (Optional parameter), a symbol of self-learning for vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;

2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;

2) The time for each self-learning is approximately 1.5s;

3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for

about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;

4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;

5) If it is found that the vibration dampening effect does not meet the requirements after using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;

6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;

7) In general, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is very obvious, it may lead to self-learning of incorrect data;

8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

1) In the following program, after executing the instruction SLDataClear All, all historical self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

SLDataClear All;

For B[0]=0,B[0]<10,Step[1]

Movj LP[0],V[30],Z[0],SLOff;

Movj LP[1],V[30],Z[CP],SLOn;

Movj LP[2],V[30],Z[CP],NWait,SLReset;

Movj LP[3],V[30],Z[0];

Set Out[1];

SLVSMODE MidLevel;

EndFor;

Note:

The instruction Jump applies to only the SCARA and Delta robots.

Since the zero point is located above in the base coordinate system of the SCARA robot, the MH may be negative in most cases.

Generally, the height parameter must satisfy $MH > \text{start point height} + LH$ and $MH > \text{end point height} + RH$.

For any abnormal setting that does not satisfy this condition, non-gate shaped motion may appear and can be used for special application.

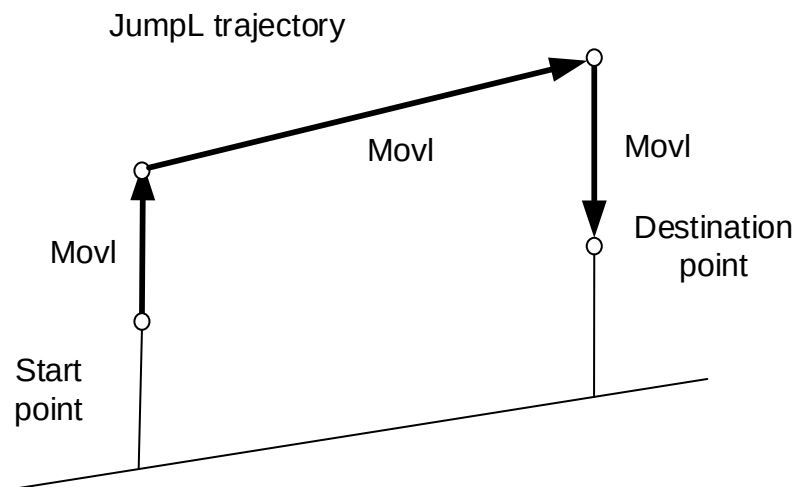
Example:

```
Jump P[1], V[30], Z[3], User[1], Tool[1], LH[60], MH[-130], RH[60];
```

3.5 JumpL

Function: Straight line jump command, with the middle section of trajectory being a straight line

Description: The motion process is divided into three sections: Movl for the rising start section, Movl for the middle section and Movl for the descending end section.



Format: JumpL P, V, Z, Tool, [User], [Acc], [Dec], [NWait], [Out(No,value,Type)...], LH, MH, RH,[SLOn/SLOff/SLReset];

Parameters:

P: Target point to reach.

Common form: P[1], LP[1], etc. It can also be in offset form and pallet point getting form.

Offset form - OffsetJ(P,PR), OffsetT(P,PR), Offset(P,PR), Offset(P,X,Y...)

In particular, PE can be used in Offset to indicate offset from the current position.

Note that if the instruction contains an offset form, there are certain requirements for the subsequent [User] and [Tool] parameters.

For details, see **Offset**.

Pallet point getting form - Pallet(PalletNo, Row, Column, Lay), LPallet(PalletNo, Row, Column, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number, see the instruction **P=PalletP=Pallet** for details.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed.

Z: Arrival and transition parameter. It describes the transition mode in the process of approaching the target point and leaving the target point.

Fine: Indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range, and then the robot performs the subsequent motion.

Z[0] to Z[200]: The robot does not stop at the target point, but smooths through the set transition area. Transition length = Transition level * Transition unit length. (To set the transition unit length,

go to Set > Motion > RunPara > Corner)

Z[CP]: The robot does not stay at the target point and smooths through the set transition area with the maximum transition length.

ZR[m]: Relative transition level, integer, value range 0 to 200, represents the percentage of maximum allowable transition length.

(1) If the current instruction is MovJ, MovL or Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half of the instruction length, which is consistent with the effect of Z[CP]. When the relative transition level m is greater than 100, the maximum allowable transition length will exceed half of the instruction. When the ZR parameter is 200, the transition length is the full length of the motion instruction.

(2) If the current instruction is Jump or Jumpl and the additional parameter RH = 0, the maximum allowable transition length will exceed half of the instruction when the relative transition level m is greater than 100. When the ZR parameter is 200, the maximum allowable transition length is the full length of the motion instruction.

(3) If the current instruction is Jump or Jumpl and the additional parameter RH≠0, the maximum allowable transition length is the total length of RH segment. At this time, ZR[100] is the total length of RH segment. When the ZR value is greater than 100, it is consistent with the effect of ZR [100].

Note:

1. When the ZR parameter is 200, the robot completely transitions the current motion to the next motion, and the robot may not pass the original trajectory at all. Because the path of motion in the transition area cannot be guaranteed, make sure that the robot transition path does not interfere with surrounding objects at low speeds.
2. Since the ZR parameter allows for a transition length longer than half the length of the instruction, in rare cases, the transition areas of the start and end points of the instruction may overlap. In this case, the robot will reduce the transition area that is more than half the size of the instruction to ensure that the two transition areas no longer overlap. In particular, if both transition areas at the beginning and end of the instruction are larger than half of the instruction, they will be reduced to half of the instruction at the same time.
3. In particular, since the transition length specified by the ZR parameter is calculated by the percentage of the maximum allowable transition length of the instruction (usually half of the instructed displacement), it is important to note that the theoretical transition length of the two instructions may be inconsistent when the displacement difference between two adjacent motion commands is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see **Transition feature**.

Tool: Tool coordinate system: Tool [0] to Tool [15].

For details, see **Extended use of Tool parameters**.

User: (Optional parameter) User coordinate system: User[0] to User[15].

For details, see **Extended use of User parameters**.

Acc: (Optional parameter) Specifies acceleration. For example, Acc[50] represents 50% of maximum acceleration.

When this parameter is not used, Acc is the same with V value by default. Minimum effective value is 20%.

Dec: (Optional parameter) Specifies deceleration. For example, Dec[50] represents 50% of maximum deceleration.

When this parameter is not used, Dec is the same with ACC value by default. Minimum effective value is 20%.

NWait: (Optional parameter) Identification of not waiting.

This item indicates that subsequent non-motion commands are immediately executed without running to a target point.

This causes subsequent non-motion commands such as operation, signal and logic judgment to be executed in advance until the next motion command or Delay command is encountered.

Description:

1. If the following instruction is a motion instruction, regardless of whether the motion instruction includes NWait, the motion instruction will be issued in advance;
2. When multiple motion instructions are used, regardless of whether the motion instruction includes NWait, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance;

Example of use:

##Set the signal immediately when the motion starts without waiting for the motion to be completed

```
JumpL P[1],V[30],Z[0],NWait,LH[60],MH[130],RH[62];
Set Out[3],ON;
```

Note for use with NWait at the end of continuous motion:

For multiple motion instructions, if the last motion instruction includes NWait followed by a non-motion instruction, the non-motion instruction will be executed in advance. Example:

001	▶ START;	001	▶ START;
002	Movj P[0],V[30],Z[0];	002	Movj P[0],V[30],Z[0];
003	Movj P[1],V[30],Z[0];	003	Movj P[1],V[30],Z[0];
004	Movj P[2],V[30],Z[0];	004	Movj P[2],V[30],Z[0];
005	Movj P[3],V[30],Z[0],NWait;	005	Movj P[3],V[30],Z[0],NWait;
006	Set Out[6],ON;	006	Set Out[6],ON;
007	END;	007	Movj P[4],V[30],Z[0];
		008	Movj P[5],V[30],Z[0];
		009	Movj P[6],V[30],Z[0],NWait;
		010	Set Out[7],ON;
		011	END;

On the left, Set Out[6] will be executed before the fifth statement. How soon it is performed in advance is affected by the program preprocessing, and in this case, it will be executed immediately following the first statement.

On the right, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

Out(No,value,Type)...: (Optional parameter) Parallel output signal during operation.

Up to two signals are output in parallel in a motion instruction.

Subparameter:

No: Output signal serial number, only 0-63 is supported.

Value: Output signal value.

Type: Type of output signals in the motion process. Different motions can use different Types.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] represents the time, unit: seconds, range -65535 to 65535. n >= 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, motion is triggered at the start or end points of this travel section at most, so the set time is invalid.	Movj/ Movl/ Movc/Jump/Jump L
Ds[n]]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when moving to n% of the path. The range is 0.000 to 100.000.	Movj/ Movl/ Movc/Jump/Jump L
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The range is -65535 to 65535. n >= 0 indicates signals are output after the robot moves to n mm from the start point. n < 0 indicates signals are output before the robot moves to n mm from the end point.	Movl/Movc/Jump L

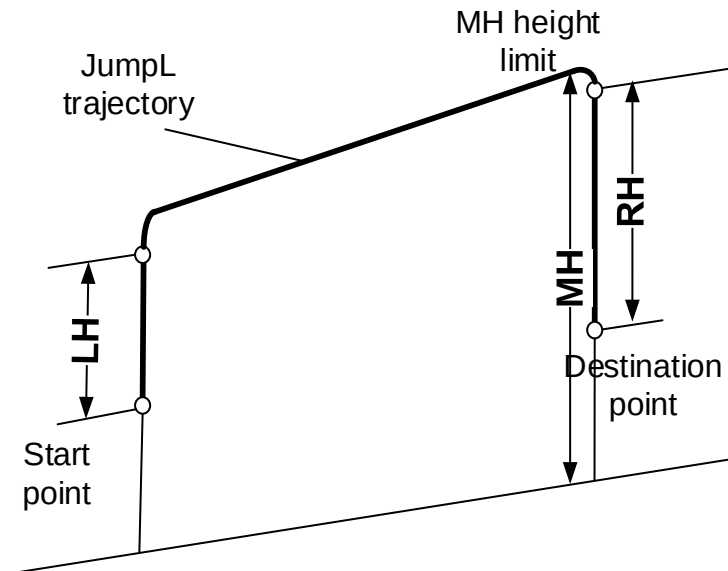
Note:

1. Precision cannot be guaranteed for motion I/O set in the transition area.
2. When T[n]/D[n]/S[n] is set beyond the motion range, motion is triggered at the start and end points of this travel section at most.
3. When Jump/JumpL uses D[n] and S[n], only the horizontal movement segment is considered.
4. When using Out (No, value, T[n]), the timer will be reset after pausing.
5. When two motion instructions with the same points are run consecutively in the program, the second motion instruction is invalid.

LH: Lifting height at the start point in a value range of 0.000 to 2000.000.

MH: It is the height of the highest point relative to the zero point of the base coordinate system in the running process. The value range is -2000.000 to 2000.000.

RH: Descending height to the end point in a value range of 0.000 to 2000.000.



SLOn/SLOff/SLReset: (Optional parameter), a symbol of self-learning for vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;

2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;

2) The time for each self-learning is approximately 1.5s;

3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;

4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;

5) If it is found that the vibration dampening effect does not meet the requirements after

using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;

6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;

7) In general, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is very obvious, it may lead to self-learning of incorrect data;

8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

1) In the following program, after executing the instruction SLDataClear All, all historical self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

```
SLDataClear All;  
For B[0]=0,B[0]<10,Step[1]  
    Movj LP[0],V[30],Z[0],SLOff;  
    Movj LP[1],V[30],Z[CP],SLOn;  
    Movj LP[2],V[30],Z[CP],NWait,SLReset;  
    Movj LP[3],V[30],Z[0];  
    Set Out[1];  
    SLVSMODE MidLevel;  
EndFor;
```

Note:

The Jump command applies to only the SCARA and Delta robots.

Since the zero point is located above in the base coordinate system of the SCARA robot, the MH may be negative in most cases.

Generally, the height parameter must satisfy $MH > \text{start point height} + LH$ and $MH > \text{end point height} + RH$. For any abnormal setting that does not satisfy this condition, non-gate shaped motion may appear and can be used for special application.

Example:

```
JumpL P[1],V[30],Z[3],User[1],Tool[1],LH[60],MH[-130],RH[62];
```

3.6 Home

Function: Returns to origin

Description: Returns to origin. You can specify the speed of motion during the returning process.

Format: Home[Index], [V] ,[SLOn/SLOff/SLReset];

Parameters:

Index: Origin number, 0 to 4

V: (Optional parameter) The speed of returning to the origin. When not set, the default speed is 30%.

SLOn/SLOff/SLReset: (Optional parameter), a symbol of self-learning for vibration suppression.

Function: It marks movements that require self-learning to suppress vibration;

Description: This parameter is the default parameter of motion instruction, marking motion actions that require vibration suppression through self-learning;

Parameters:

1) This parameter is the parameter of the motion instruction, including three parameters SLOn, SLOff and SLReset. SLOn indicates that this segment of motion requires self-learning, SLOff indicates that this segment of motion does not require self-learning, and SLReset indicates that this segment of motion requires re-learning;

2) The default is SLOff, which means that self-learning is not required for this segment of motion;

Detailed description:

1) Self-learning is possible only when the global motion rate is greater than 49%, and not when it is less than 49%;

2) The time for each self-learning is approximately 1.5s;

3) If the parameter is SLOn, when the instruction is first executed, the robot will stop for about 1.5s after it reaches the destination, automatically learning the relevant information and saving it in the learning data file. Under normal circumstances, the relevant information required can be learned automatically after one run, so the robot will not need to stop again to learn when called repeatedly later;

4) If the parameter is SLReset, the robot must stop for about 1.5s after each motion reaches the destination, automatically learning relevant information and saving it in the learning data file;

5) If it is found that the vibration dampening effect does not meet the requirements after using the SLOn parameter for self-learning, the SLReset parameter can be used, which is equivalent to enforcing self-learning every time the motion is performed. Generally, it is only used when commissioning. When it is confirmed that the vibration meets the requirements, the SLReset needs to be changed to SLOn or SLOff;

6) For motion segments with good vibration, it is best not to mark them (default or SLOff), otherwise the robot may stop for self-learning when it first runs to the marked position;

7) In general, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is very obvious, it may lead to self-learning of incorrect data;

8) For motion instruction with SLOn, SLReset parameters, it is necessary to wait for the execution of this instruction before executing subsequent instructions, so the transition parameter and the NWait parameter of this segment of the motion will be forced not to take effect;

Example:

1) In the following program, after executing the instruction SLDataClear All, all historical self-learning data will be cleared. Therefore, regardless of whether self-learning has been performed before, when moving to LP [1] and LP [2] points, self-learning will be performed. The motion to the

LP[2] point continues self-learning before jumping out of the for loop. Since the self-learning parameters are SLOff or default, self-learning will not occur when moving to LP [0] and LP [3] points. Due to the presence of the SLOn and SLReset parameters, neither of the two motions to the LP[1] and LP[2] points will have a transition effect;

```
SLDataClear All;
  For B[0]=0,B[0]<10,Step[1]
    Movj LP[0],V[30],Z[0],SLOff;
    Movj LP[1],V[30],Z[CP],SLOn;
    Movj LP[2],V[30],Z[CP],NWait,SLReset;
    Movj LP[3],V[30],Z[0];
    Set Out[1];
    SLVSMODE MidLevel;
  EndFor;
```

Example:

```
Home[2]V[30];          ##Returns to origin 2 (i.e., the third origin) at 30% maximum speed
```

3.7 ArmChange

Function: Switches arms of SCARA robot

Description: Switches between left and right arms of the SCARA robot

Format: ArmChange value,V;

Parameters:

Value: Arm position, can be -1, 1, 0.

- -1 represents the left arm.
- 1 represents the right arm.
- 0 stands for automatic switching, changing to the arm opposite to the current arm.

V: Switch speed, e.g. V[50] represents 50% of the maximum speed.

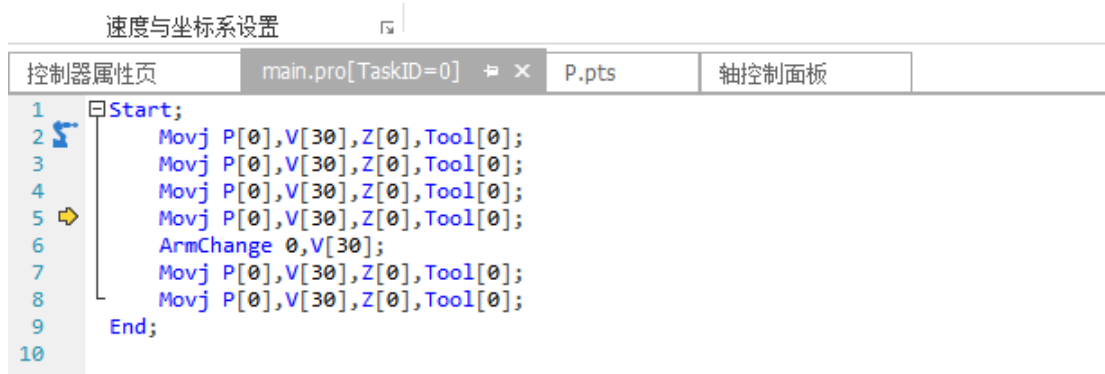
Note:

1. The transition of ArmChange is Z [0].
2. When the SCARA robot is near a singularity (the angle of the J2 axis is less than 0.5°), it cannot switch arms, otherwise an error will be reported.

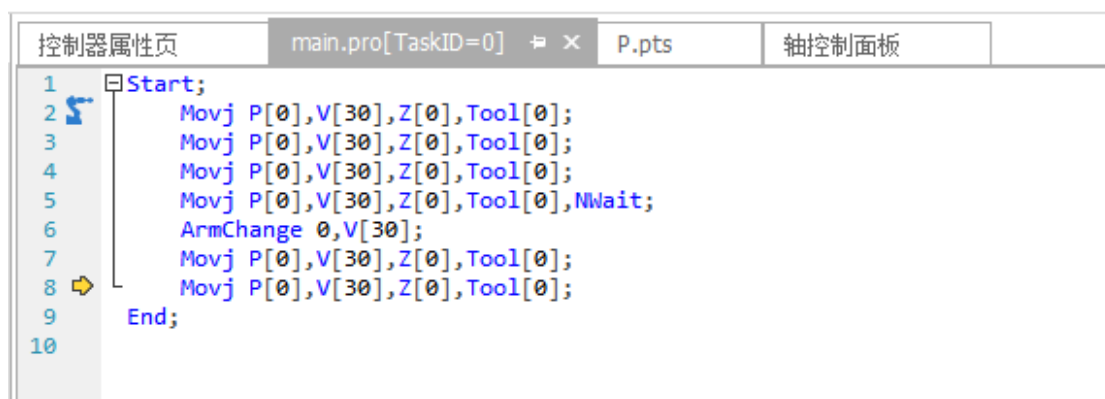
Example:

```
ArmChange 1,V[20];
```

3. When the motion instruction preceding the instruction ArmChange does not include Nwait, the instruction ArmChange is not precompiled, meaning that there is no transition for the instruction ArmChange in this case. In the program shown in the following figure, it will only be pre-compiled to line 5.



If you want to precompile the instruction Armchange, you can add the Nwait flag to the preceding instruction to achieve the transition to Armchange. In the program shown in the following figure, it will only be pre-compiled to line 8.



4 Signal Processing Instructions

4.1 Set

Format 1: Set Out

Function: Output

Description: Sets a single digital output signal

Format: Set Out,value;

Parameters:

Out: Digital output signal, Out[0] to Out[13823]

Value: The value of digital signal can only be ON/OFF.

Example:

Set Out[1], #Sets Out[1] to ON

Format 2: Set OG

Function: OG output

Description: Sets a group of digital output signals

Format: Set OG, BValue ;

Parameters:

OG: Outputs a group of signals, OG[0] to OG[15]; elements defined by OG[Index] are defined by the instruction Group

BValue: Values or variables of Byte type, range 0 to 255, representing a set of signal states

Description:

1. Before using the OG, you must use the instruction Group to define the corresponding I/O element of the OG, otherwise it will report an error that the OG is not defined. 2. The value of the variable is an integer between 0 and 255, which is exactly an 8-bit binary number, each bit from right to left represents a signal state from low to high.

Example:

Group OG[0],0,1,2,3,4,5,6,7;

B[1]=7;

Set OG[0],B[1]; ##Sets OG[0] output to 0000 0111, i.e. Out[0] to Out[7] are 1110 0000 respectively.

Format 3: Set DA

Function: Sets DA

Description: Sets analog output signals

Format: Set DA,Value;

Parameters:

DA: Analog output signal, DA[0] to DA[15]

Value: If the signal is current, the default unit is mA; if the signal is voltage, the default unit is V.

Identify whether it is current or voltage based on the configuration of the IRLink module. Limit the inputs based on the configured current or voltage range.

Example:

Set DA[1], 1.2; ##If DA[1] is configured as the current type, this instruction outputs a 1.2 mA current.

Note:

When editing instructions, if the values in the instructions exceed the actual configured range, there will be restrictions or prompts. If there are I/O instructions in the pre-programmed program and the configuration of the IRLink module is subsequently modified, only the boundary values of the actual configuration will be taken.

4.2 Get

Format 1: Get In

Function: Reads a single digital input signal

Description: Reads a single digital input signal

Format: Get In, Var;

Parameters:

In: Digital input signal, In[0] to In[13823]

Var: variable, saves the read results. 0 for OFF, 1 for ON.

Example:

```

START;
Get In[7],B[1];
Switch B[1]
    Case 0:
        Print "In[7] is OFF";
        Break;
    Case 1:
        Print "In[7] is ON";
        Break;
    Default:
        Print "Error";
EndSwitch;
END;

```

Format 2: Get Out

Function: Reads a single digital output signal

Description: Reads a single digital output signal

Format: Get Out,Var;

Parameters:

Out: Digital output signal, Out[0] to Out[13823]

Var: Variable, saves the read results. 0 for OFF, 1 for ON.

Example:

```

START;
Get Out[0],B[1];
Switch B[1]
    Case 0:
        Print "Out[0] is OFF";
        Break;
    Case 1:
        Print "Out[0] is ON";
        Break;
    Default:
        Print "Error";
EndSwitch;
END;

```

Format 3: Get IG

Function: Reads a group of digital input signals

Description: Reads a group of digital input signals

Format: Get IG, Var;

Parameters:

IG: Input signal group, range IG[0] to IG[15] , elements defined by IG[Index] are defined by the instruction Group

Var: Variable, saves the read results. It represents a set of signal states.

Description:

1. Before using the OG, you must use the instruction Group to define the corresponding I/O element of the OG, otherwise it will report an error that the OG is not defined.
2. The value of the variable is an integer between 0 and 255, which is exactly an 8-bit binary number, each bit from right to left represents a signal state from low to high.

Example:

```
Group IG[1]0,1,2,3,4,5,6,7;##Defines IG group
```

```
Get IG[1],B[1];
```

```
##Reads the signal of IG[1];
```

```
##If signals In[15] to In[8] are OFF, OFF, OFF, OFF, OFF, ON, ON, ON successively, then  
B[1]=7.
```

Format 4: Get OG

Function: Reads a group of digital output signals

Description: Reads a group of digital output signals

Format: Get OG[***], Var;

Parameters:

OG: Outputs a group of signals, OG[0] to OG[15]; elements defined by OG[Index] are defined by the instruction Group

Var: Variable, saves the read results. It represents a set of signal states.

Description:

1. Before using the OG, you must use the instruction Group to define the corresponding I/O element of the OG, otherwise it will report an error that the OG is not defined.
2. The value of the variable is an integer between 0 and 255, which is exactly an 8-bit binary number, each bit from right to left represents a signal state from low to high.

Example:

```
Group OG[1]0,1,2,3,4,5,6,7;##Defines OG group
```

```
Get OG [1],B[1];
```

```
##Reads the signal of OG[1];
```

```
##If signals Out[15] to Out[8] are OFF, OFF, OFF, OFF, OFF, ON, ON, ON successively,  
then B[1]=7.
```

Format 5: Get AD

Function: Reads analog input signal

Description: Reads analog input signal

Format: Get AD,Var;

Parameters:

AD: Analog input signal, AD[0] to AD[15]

Var: Variable, saves the results. If the signal is current, the default unit is mA; if the signal is voltage, the default unit is V.

Automatically identify whether it is current or voltage based on the configuration of the IRLink module.

Example:

Get AD[1],D[1]; ##Gets value of AD[1]

4.3 Wait

Function: The instruction Wait is used to wait for the establishment of an event or time, otherwise the task remains in the wait state.

Description: Wait until the input and output signals are detected to meet the conditions.

Format:

Wait Condition inequality;

Description: Wait until the conditional inequality holds before executing the next line of instruction

Wait T[Time];

Description: Wait for the specified time before executing the next line of instruction

Wait Conditional inequality, T[time];

Description: Wait until the conditional inequality holds within the specified time before executing the next line of instruction, otherwise an alarm will be triggered after timeout

Time: Time value, double data, 0 to 65535. Unit: second, Time=0 means that only one judgment is made.

Wait Conditional inequality, T[time], Goto L[***];

Description: Wait until the conditional inequality holds within the specified time before executing the next line of instruction, otherwise jump to the line where the label L[i] is located after the timeout. Time: Time value, double data, 0 to 65535. Unit: second, Time=0 means that only one judgment is made.

Parameters (supported variables):

Conditional inequality Numeric variable/Numeric value/Numeric expression $</>$ $<=>$ $=/=$ $<>$

Numeric Variable/Variable/ Numeric expression

Numeric expression

Inequality sign Numeric expression

T[Time] (optional): T range 0.00 to 65535.00s

Goto L[***] (optional): Jump to the line where the label is located

Example:

Wait In[6] == ON,T[10];

Description: Wait until the signal of input port [6] is ON before executing the subsequent instructions; wait up to 10 seconds, and if the condition is still not met after 10s, an alarm will be generated.

Wait B[2]>5,T[2],Goto L[1];

##Executes the subsequent program

L[1]:

##Error handling

Description: Wait until the value of the B[2] variable is greater than 5 before executing the subsequent instructions; wait up to 2s, and if the condition is still not met after 2s, jump to L[1].

Note:

1. The time to wait again after canceling the pause is the total time minus the time consumed

```

before the pause, for example,
Wait B[2]>5,T[2],Goto L[1];
##Executes the subsequent program
L[1]:
##Error handling

```

Description: Wait until the value of B[2] variable is greater than 5 before running the subsequent instruction; wait up to 2s.

If wait for 1s and pause, start again and the condition is still not met after 1s, then jump to L[1].

4.4 Group

Function: Configures group signal

Description: Configures input and output group signals. A group signal contains up to 8 signals. Group I/O outputs can be defined using the instruction Group. The number of signals in a group can be smaller than 8. Note that IG and OG must be defined using the instruction Group before they can be used. Once the instruction Group is executed, as long as the controller is powered on again, the elements in the IG and OG will be cleared. In other cases, as long as the instruction Group is executed, the corresponding IG or OG will be recombined. If the instruction Group is not used, the default IG and OG arrays will be empty, and using IG and OG arrays in the program will result in an error.

Format:

```

Group OG, n0, n1, n2..., n7;
Group IG, n0, n1, n2..., n7;

```

Parameters:

OG: Output group to be redefined, OG[0] to OG[15]. Up to 16 groups can be defined.

IG: Input group to be redefined, IG[0] to IG[15]. Up to 16 groups can be defined.

n0, n1, n2..., n7: Serial number of signals. You can configure a maximum of 8 signals, and a minimum of 2 signals.

Scope: Effective in the system. Note that once the instruction Group is executed, it remains in effect. If the instruction Group has been executed in a previous project, it can be used directly in a new project without any errors.

Default restore: Restore to undefined state on re-powering.

Example:

Group OG[0] 1,2,7; ##Redefines OG[0] as Out[1], Out[2] and Out[7].

4.5 Invert

Function: Inverts signal

Description: Invert output signal

Format: Invert Out;

Parameters:

Out: Output signal, Out[0] to Out[13823]

Example:

Set Out[1],ON;

Invert Out[1]; ##Out[1] is OFF

5 System Parameters-Related Instructions

5.1 SetAccRamp

Description: Sets the jerk of the motion segment

Format: SetAccRamp(StartValue,EndValue);

Parameters:

StartValue: Jerk percentage of the start segment. Integer data, range (1 to 100).

EndValue: Deceleration percentage of the end segment. Integer data, range (1 to 100).

Note:

- 1) This command is effective for Movj, Movl, Movc, Jump, JumpL, ArmChange, and Home motions and will affect the jerk in these motions.
- 2) When the instruction SetAccRamp is not used, the system default jerk parameter is dependent on the model.
- 3) The scope of SetAccRamp is the whole system. For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)
- 4) The parameters input by SetAccRamp are strongly related to vibration, and arbitrary adjustment may cause the robot to vibrate more severely. Please adjust the input parameters appropriately according to actual needs and the robot's vibration conditions.

5.2 SlewMode

Function: Set the rotation optimization mode

Description: Sets the rotation optimization mode for the J4 axis of the SCARA robot or the J6 axis of the standard 6-axis robot.

Format: SlewMode nMode;

Parameters:

nMode: Rotation optimization modes 0, 1, 2, 3.

(For standard 6-axis robots, replace J4 below with J6)

0: No optimization, arm parameters in position variables prevail

1: Optimization mode 1 to ensure that J4 is always within the range of -180 to 180 degree during motion.

2: Optimization mode 2 to ensure that J4 always moves in the shortest way during motion. That is, to measure whether the target point to move to requires J4 to rotate by 180 degrees. If the angle difference is $\leq 180^\circ$, the target point is fully reached. If $> 180^\circ$, then J4 moves in the opposite direction and ultimately moves to a position where J4 is 360° away from the target point. If the reverse movement of J4 exceeds the limit range of the robot, it will stop and alarm.

3: Optimization mode 3, similar to mode 2 in which J4 moves in the shortest way. The difference is that if the reverse motion of J4 exceeds the limit range of the robot, there will be no alarm, but instead no reverse motion will be carried out and the original normal motion will be fully adopted.

Example:

.....

SlewMode 3; ##Adopts optimization mode 3

Movj P[1],V[30],Z[0],Tool[1];

Movj P[2],V[30],Z[0],Tool[1];

.....

SlewMode 0; ##Optimization off

.....

Note: Once this instruction is set, it remains in effect for the motion after the instruction, unless it is set again.

For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)

5.3 Velset

Function: Sets speed

Description: There are two ways to set the global speed: Velset Rate[value] and Velset value, one to set the global speed percentage and the other to force the instruction speed. For Velset value, once set, this instruction remains in effect until it encounters a Velset OFF or is replaced by another setting.

**Operating speed = set maximum speed * global speed
percentage * percentage set by instruction**

**V in motion instruction
Affected by Velset value
Remove effect by Velset OFF**

**Speed percentage set in toolbar
Affected by Velset Rate[value]**

Format:

Format 1: Velset Rate[value];

Parameters:

Rate[value]: Global speed percentage.

Description:

Sets the global speed percentage. When you run this command, the global speed percentage on the toolbar changes.

Format 2: Velset value;

Parameters:

Value: Forces the instruction speed value as a percentage.

Description:

Ignore the V parameter after the Move and Jump series instructions, and always use Value as the percentage set by the instruction.

Note: VelSet value is very special and is not affected by the program logic, it is a macro definition processing. That is, as long as the instruction exists, even if it is in an If-else that is not executed, it

will still affect the instructions that come "after" it. Here "later" means later in the program line, not later in the actual logical order of execution.

Scope: Between the Velset value in the current program file and Velset OFF.

Format 3: Velset OFF;

Cancels the speed setting of format 2 "Velset value". Leaves the speed settings in the motion instruction in effect.

Example:

##Assume the current speed is 100% on the toolbar

START;

Movj P[0],V[70],Z[3],Tool[1]; ##Actual speed is 100%*70%=70%

Velset Rate [50];

Movj P[1],V[70],Z[3],Tool[1]; ##Actual speed is 50%*70%=35%

Velset 30;

Movj P[2],V[70],Z[3],Tool[1]; ##Actual speed is 50%*30%=15%

Sub1.Func1(); ## Data in the subroutine is not affected by Velset 30

Movj P[3],V[70],Z[3],Tool[1]; ##Actual speed is 50%*30%=15%

Velset OFF;

END;

5.4 GetAlarmNo

Description: Gets alarm number and saves value in specified variable

Format: GetAlarmNo(Var);

Parameters:

Var: Numeric variable (int type) used to store the acquired alarm number. The alarm number is stored in decimal and needs to be converted to hexadecimal before it can be queried against the alarm list (Appendix 1).

Example:

GetAlarmNo(R[1]);

Note:

Pay attention to the numerical range of the variable. For example, if a B variable is used, the alarm number returned may exceed the range and cause an alarm. If a custom Byte variable is used, it will be forcibly converted to the maximum value for that type.

5.5 GetRunState

Description: Gets the system operating status and stores the value in the specified variable

Format: GetRunState(Var);

Parameters:

Var: Numeric variable (integer type) used to store system operation status.

0- Stop

1- Start

2- Single-step forward

Example:

GetRunState(Var);

```

Switch B[1]
  Case 0:
    Print "State is stop";
    Break;
  Case 1:
    Print "State is run";
    Break;
  Case 0:
    Print "State is step";
    Break;
  Default:
    Print "State is undefine";
    Break;
EndSwitch;

```

5.6 SetSysToolNo

Description: Set the tool number used by the system

Format: SetSysToolNo(ToolNo);

Parameters: ToolNo: Tool number, integer, range 0 to 15

Example

```
SetSysToolNo(4);
```

##As a result, the tool number used by the system in the upper right corner of the teach pendant has changed to 4.

5.7 SetSysUserNo

Description: Sets the user number used by the system

Format: SetSysUserNo(UserNo)

Parameter:

UserNo: User number, integer value, range 0 to 15

Example

```
SetSysUserNo(4)
```

##As a result, the user number used by the system in the upper right corner of the teach pendant has changed to 4.

5.8 SetFlyMode

Description: Sets the transition mode for motion instructions

Format: SetFlyMode (MotionType,Mode);

Parameters:

MotionType: Motion type. Currently, only CP is supported and is only valid in motion types such as Movl, Movc, and Jumpl.

Mode:

FLY_FREE: Free transition mode. The transition path is automatically selected based on the motion status of the previous and next motion instructions. The free transition mode usually has a better transition effect, but its motion path in the transition area may vary with the motion parameters in the system or instruction, and is suitable for applications such as handling and sorting that do not require a fixed path in the transition area.

FLY_FIX: Fixed path transition mode. The robot transitions according to a fixed motion path, and the motion path of the robot in the transition area does not change with speed or acceleration. It is suitable for small-segment machining or trajectory applications that require the transition path not to vary with speed. To balance efficiency and motion smoothness, the robot's transition speed may be constrained by preset position and pose transition stresses. To adjust the transition stress, use [the SetFlyPress](#) transition stress setting instruction. (Note S03.21 controller version does not support fixed path transition mode, and the robot will ignore this setting and transition according to FLY-FREE free transition mode.)

Note:

1. When the motion type is set to CP, this instruction is valid for Movl, Movc, JumpL motions and affects the transition mode between these motions.
2. In free transition mode, when the motion parameters in the robot system or instructions change, the motion path of the robot within the transition region may change, and the changed path remains within the given transition area. If you want the transition path to remain the same, set the transition mode to fixed path mode.
3. The system defaults to free transition mode when the instruction SetFlyMode is not used.
4. The scope of SetFlyMode is the whole system. When using the instruction SetFlyMode to set the transition mode in the system, the system's transition mode will be updated. For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)

5.9 SetFlyPress

Description: Sets the transition stress for fixed path transition

Format: SetFlyPress (PosPress, OrientPress);

Parameters:

PosPress: position transition stress. Integer data, range (50 to 200). The default value is 100%.

OrientPress: Posture transition stress. Integer data, range (50 to 200). The default value is 100%.

Note:

- 1) This instruction is valid for the fixed path transition segment (FLY_FIX) of the CP motion instructions (Movl, Movc, JumpL) and affects the motion speed of the fixed path transition segment.
- 2) When the instruction SetFlyPress is not used for transition stress setting, the system defaults to transition at 100% positional transition stress and 100% pose transition stress.
- 3) When making a fixed path transition, the robot's speed of motion is usually limited by position transition stress and pose transition stress. In general, increasing the position or pose transition stress may increase the speed of motion of the robot in the transition segment, but may at the same time exacerbate the impact during the motion. Be careful to avoid excessive transition stress to prevent vibration.

4) The scope of SetFlyPress is the whole system. When using the instruction SetFlyPress to set transition stress in the system, the system transition stress will be updated. For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)

5.10 SetToolParm

Function: Sets tool coordinate system

Description: Set the tool coordinate system parameters dynamically, either by the direct method or by the three-point method TCP, as follows in format 1 and format 2. Once set, the tool coordinate system values are changed immediately in the program until the change is turned off using SetToolParm (ToolNo,OFF) and the original values are restored.

Format 1: SetToolParm (ToolNo,PR);

Description: Sets the value of the tool coordinate system directly

Parameters:

ToolNo: Tool number, value range 1 to 15

PR: Pan variable, used to input the parameter values of the tool coordinate system to be set.

Format 2: SetToolParm (ToolNo,P[i],P[j],P[k]);

Description: Calculates and sets the tool coordinate system using the three-point TCP method. (See three-point method TCP for tool coordinates in Section 3.2.3(a))

Parameters:

ToolNo: Tool number, value range 1 to 15

P[i],P[j],P[k]: Calculates the three points used in the tool coordinate system by three point method TCP.

Format 3: SetToolParm (ToolNo,OFF);

Description:

Restores the tool coordinate system parameters to the original values.

Parameters:

ToolNo: Tool number, value range 1 to 15

Note:

P[i]P[j]P[k] must be defined in advance if format 2 is used.

The coordinate system values set are used temporarily only in the program and the values displayed on the coordinate setting page do not change. After use, use SetToolParm (tool number, OFF) to restore it.

Scope of instruction SetToolParm: Only valid for the current project. When the project is re downloaded and recompiled, it is restored to the initialized values in the project file.

5.11 SetUserParm

Function: Sets user coordinate system

Description: Dynamically sets the user coordinate system parameters, either by direct method or three-point method, two-point method, in the following formats 1, 2, and 3. Once set, the user

coordinate system values change immediately in the program until the change is turned off using SetUserParm (UserNo, OFF) to restore the original values.

Format 1: SetUserParm (UserNo,PR);

Description: Sets the value of the user coordinate system directly

Parameters:

UserNo: User number, value range 1 to 15

PR: Used to enter the value of the tool coordinate system parameters.

Format 2: SetUserParm (UserNo,P[i],P[j],P[k]);

Description: Calculates and sets the user coordinate system using the three-point method. (See three-point method for user coordinates in Section 3.2.3(b))

Parameters:

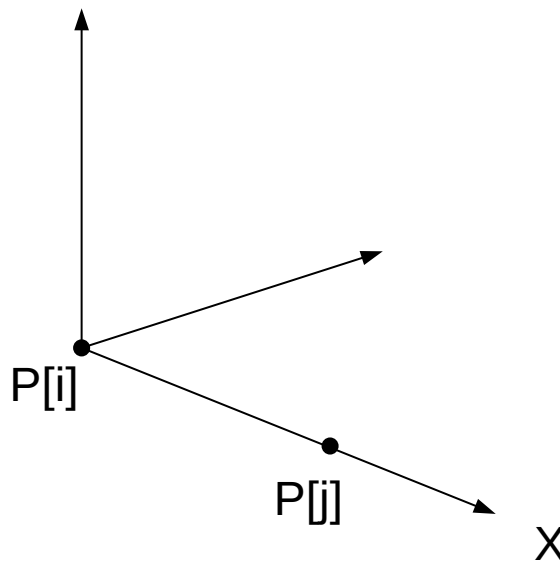
UserNo: User number, value range 1 to 15

P[i],P[j],P[k]: Calculates the three points used in the user coordinate system by the three-point method.

Format 3: SetUserParm (user number,P[i],P[j]);

Description: Calculates and sets the user coordinate system using the two-point method. The two-point method here uses P[i] as the origin, and the Z-axis of the user coordinate system is vertically upright by default, so only two points P[i], P[j] are required to determine the X-axis.

Z (Straight upwards)



direction.

Parameters:

User number: Can be a numeric value 1 to 15 or a B/LB variable.

P[i],P[j] is a position variable.

Format 4: SetUserParm (UserNo,OFF);

Description:

Restores the original user coordinate system parameters to the original values.

Parameters:

UserNo: User number, value range 1 to 15

Note:

If P[i],P[j],P[k] are used, these parameters must be defined in advance.

After use, use SetUserParm (ToolNo, OFF) to restore it.

The coordinate system values set are used temporarily only in the program and the values displayed on the coordinate setting page do not change.

Scope of instruction SetUserParm: Only valid for the current project. When the project is re downloaded and recompiled, it is restored to the initialized values in the project file.

5.12 OffSetUserParm

Function: Offsets the user coordinate system

Description: Offsets the user coordinate system according to the offset variables in the direction of the base coordinate system. Once set, the user coordinate system values change immediately in the program until the change is turned off using OffSetUserParm (UserNo, OFF) to restore the original values.

Format 1: OffSetUserParm (UserNo,PR);

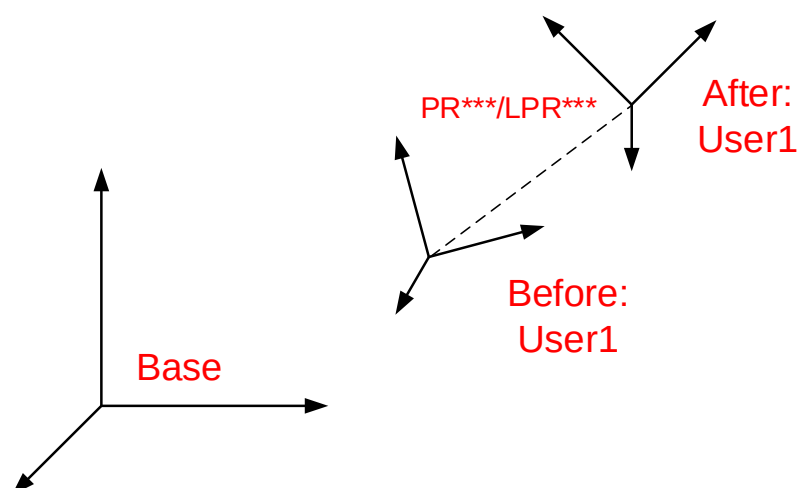
Description: Offsets the user coordinate system, and the offset amount is stored in the PR/LPR variable.

Parameters:

UserNo: User coordinate system number

PR: Offset variable for offset based on the base coordinate system

Illustration:



Format 2: OffSetUserParm (UserNo,OFF);

Description: Turns off the offset of the user coordinate system

Parameters:

UserNo: User coordinate system number

OFF: Turns off the offset of the user coordinate system to restore the user coordinate system parameters

Note:

The coordinate system values set are used temporarily only in the program and the values displayed on the coordinate setting page do not change.

After use, use OffSetUserParm (Tool Number, OFF) to restore it.

Scope of instruction OffSetUserParm: Only valid for the current project. When the project is re downloaded and recompiled, it is restored to the initialized values in the project file.

Example:

PR1=(20,30,0,0,0,0);

OffsetUserParm(2,PR1);

5.13 Clear

Description: Clears the value of the object. The object can be a numeric variable, a translation variable, a tool coordinate system, a user coordinate system, a handheld work load, or an Out signal. You can specify the number of objects to achieve batch clearing, clearing multiple consecutive variables starting from the selected object.

Format 1: Clear object, Num;

Parameters:

Object: Can be any of the following:

Tool	Clears tool coordinate system parameters, including tool load parameters, Tool[0] to Tool[15]
User	Clears user coordinate system parameters, User[0] to User[15]
Obj	Clears handheld workload parameters, Obj[0] to Obj[15]
Out	Clears Out variables, Out[0] to Out[13823]
Fin	Clears the state of the corresponding In variable as not mandatory, only Fin[0] to Fin[63] are supported
Box	Clears the interference area configuration in the instruction, Box[0] to Box[15]
B/LB	Clears B/LB variables
R/LR	Clears R/LR variables
D/LD	Clears D/LD variables
PR/LPR	Clears PR/ LPR variables
Alarm	Clears alarms (used in static tasks)
Port	Clears port data

Num: Number of items to clear. Clears multiple variables consecutively from Object.

Format 2: Clear All

Clears all variables, including all the above Tools, User, Obj, B, R, D, and PR variables, and also clears alarms and all open port data, excluding Pin and Out.

Description:

Tool/User/Obj clearing is to restore the variable values set in the system parameters, not zeroing the object values. In the program, the instructions SetToolParm, SetUserParm, SetToolMass, SetToolCog, SetToolOrient, SetToolInertia, SetGripLoadMass, SetGripLoadCog, SetGripLoadOrient, SetGripLoadInertia are used to change the tool, user, and work object parameters, and you can restore the original values through the instruction Clear.

Out clearing is to set the Out signal to OFF.

Fin clearing is to set the In signal to an unforced state.

B/R/D clearing is to set the value to 0.

Port clearing is to clear the port data.

PR clearing is to set all parameters in the PR variable to 0, i.e. (0,0,0,0,0,0).

Clear Alarm: Used in static tasks to clear alarms.

If an alarm is triggered during operation, it will cause the main task and non-static task to stop. In this case, the instruction Clear Alarm in the main task or non-static task will not be executed, so it will not work to clear the alarm.

Example

```
Clear Tool[1],2;
```

```
Clear Out[4],3;
```

```
Clear PR[2],4;
```

5.14 SetFlyWait

Function: Turns on or off the transition function for non-motion instructions

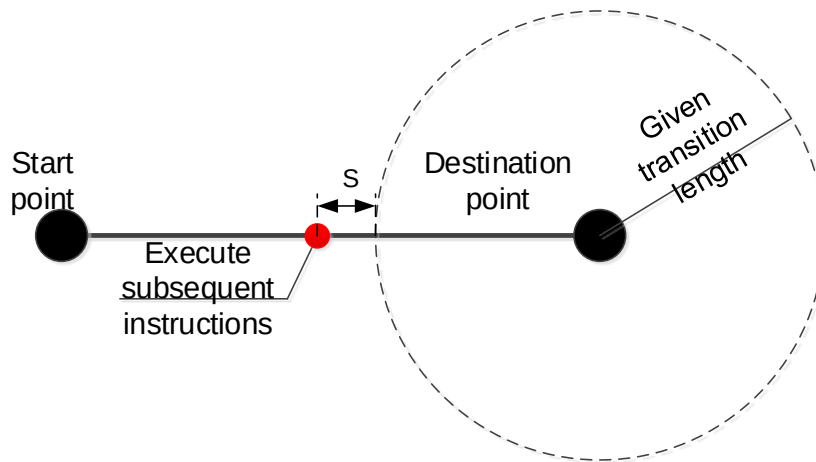
Format: SetFlyWait (ON/OFF);

Parameters: OFF: Turns off transition wait mode (default), ON: Turns on transition wait mode

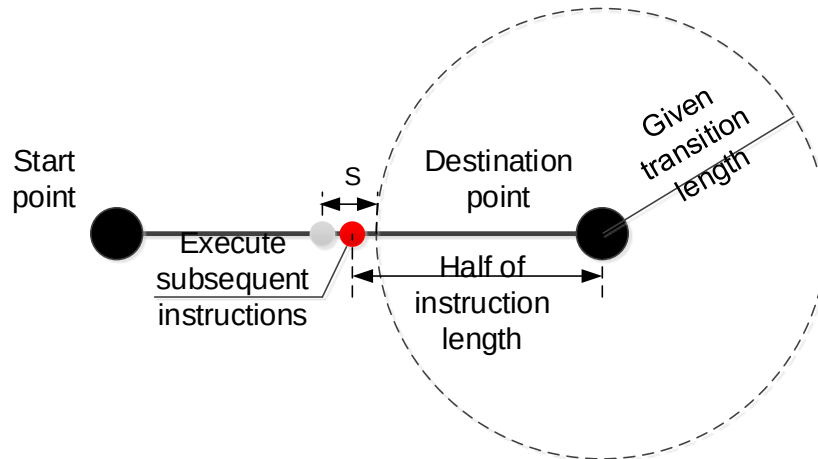
Note:

OFF: Turns off transition wait mode (default):

By default, the transition wait mode is turned off. When this setting is applied, the robot directly executes subsequent non-motion instructions at the distance S before entering the transition area. Generally, the distance S is 0.1 times the unit transition length (typical value: 1 mm or 0.3 degrees. For setting of the unit transition length, see the transition setting.)

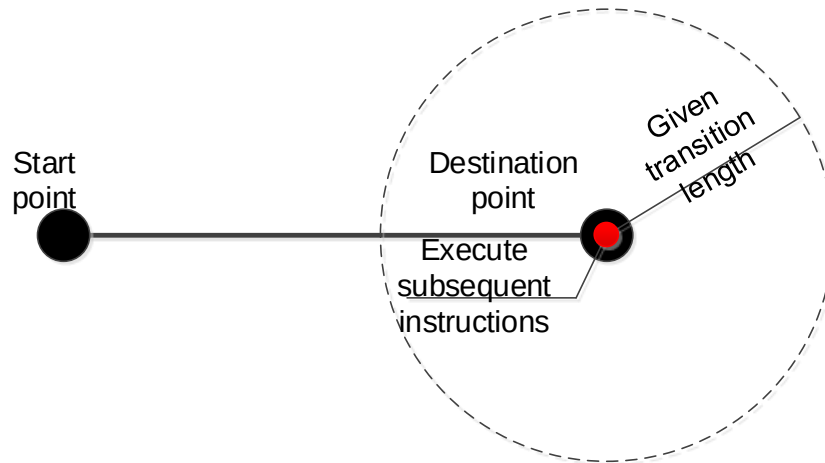


When the transition length of the motion instruction + distance S is greater than half the length of the instruction, the robot will execute subsequent non-motion instructions as soon as it reaches half the instruction. In particular, if the current motion instruction is a Jump or JumpL instruction and has a vertical descent segment ($RH \neq 0$), the robot will execute subsequent non-motion instructions as soon as the vertical descent begins.

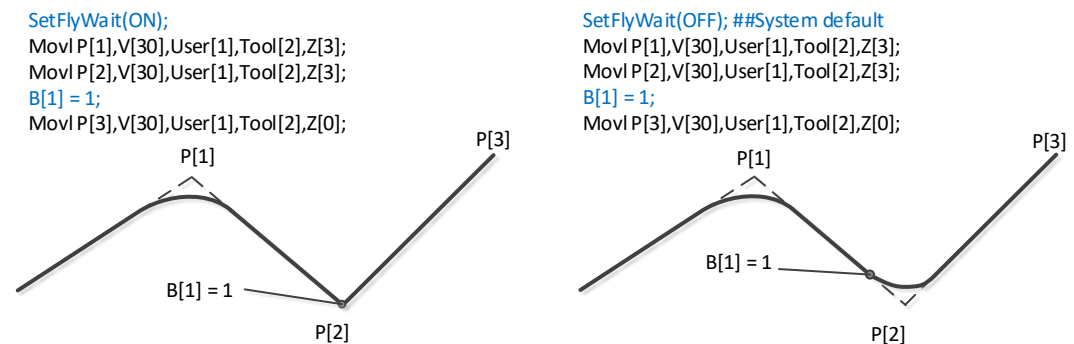


ON: Turns on transition wait mode:

When transitional wait mode is enabled, the robot executes subsequent non-motion instructions after motion has stopped. When this setting is applied, the timing of instruction execution of the robot is consistent with previous versions. That is, when a non-motion instruction exists between two adjacent motion instructions, the robot will decelerate and stop at the target point before executing subsequent non-motion instructions.



Comparison of running effect when transition wait mode is enabled or disabled



Example of running with transition wait mode OFF:

Example 1:

MovJ P[1], V[100], Z[3], Tool[1];

Set Out[4],On

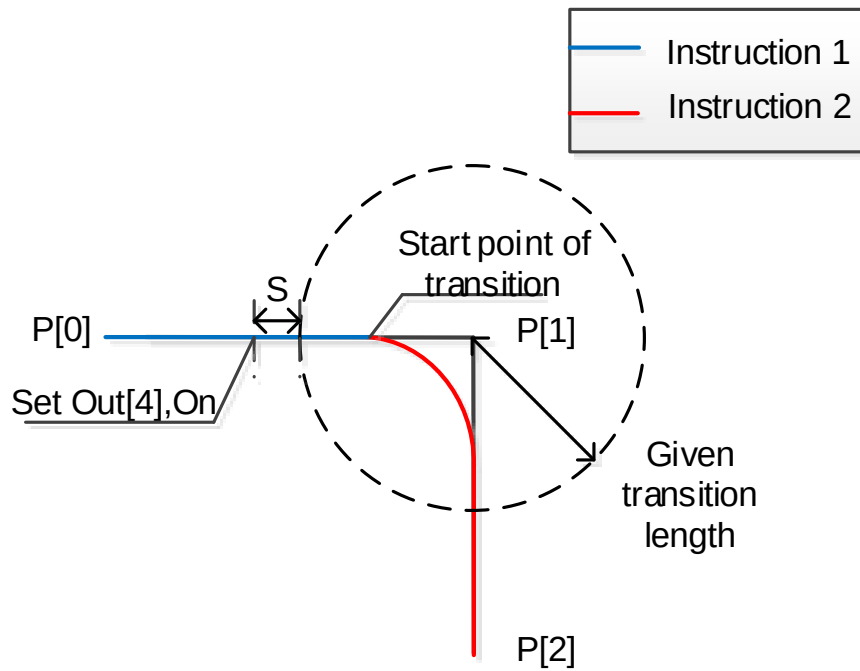
MovJ P[2],V[100],Z[0],Tool[1];

Resolution:

Move from current position P[0] to P[1]

Execute Set Out[4]On at distance S before entering transition area Z[3]

Transition occurs when passing P[1] point.



Example 2:

MovJ P[1], V[100], Z[CP], Tool[1]

Set Out[4],On

MovJ P[2], V[100], Z[0], Tool[1];

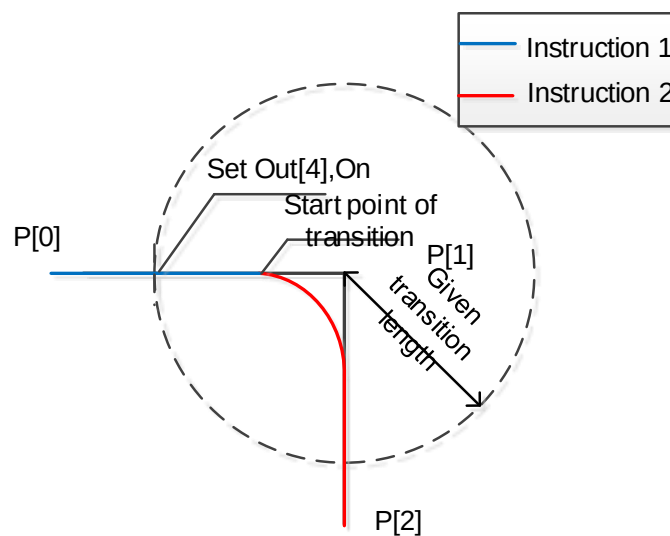
Resolution:

Move from current position $P[0]$ to $P[1]$

Since the transition length has reached half of the instruction length, the lead S is reduced to 0,

and the robot executes Set Out[4]On after reaching 50% of displacement from $P[0]$ to $P[1]$;

Transition occurs when passing $P[1]$ point, with a transition area slightly less than 50%.



Example of running when transitional waiting is ON:

SetFlyWait(On)

MovJ P[1], V[100], Z[CP], Tool[1];

Set Out[4], On

MovJ P[2], V[100], Z[0], Tool[1];

Resolution:

Move from current position P[0] to P[1]

After reaching P[1] point, the robot slows down and stops and executes Set Out[4]On

Move from P[1] to P[2].

Note:

- (1) This feature is not supported in dynamic coordinate systems.
- (2) When a subsequent instruction is one of the following, the robot waits for motion to stop before executing the instruction.
 - (1) Pause
 - (2) END
- (3) The feature of transition without waiting is only supported in free transition mode (default). When the user switches the instruction transition mode to fixed path transition mode via the SetFlyMode instruction, the non-motion instructions following the CP motion instructions (Movl, Move, JumpL) need to wait for the robot to slow down and stop before being executed. (See the instruction SetFlyMode for the explanation and related settings of the transition mode).
- (4) In general, when a given transition length is greater than half of the current motion instruction length, the robot starts executing subsequent non-motion instructions when the motion of the instruction reaches half. In particular, when the transition level is defined using the ZR parameter and the percentage of the specified transition length exceeds 100, the robot will execute subsequent non-motion instructions as soon as it reaches half of the instruction displacement.
- (5) If the current instruction is a Jump/JumpL instruction with a vertical descent segment ($RH \neq 0$), the robot will execute subsequent non-motion instructions no earlier than the vertical descent begins.
- (6) This function controls the timing of the execution of subsequent non-motion instructions only if the current motion instruction does not have the NWait parameter enabled. If the current motion is accompanied by the NWait parameter, the robot will likely execute subsequent non-motion instructions before the motion starts, even in the SetFlyWait (ON) state. (See the list of additional parameters for each motion instruction for how to use the NWait parameter).

- (7) Depending on the computational load of the robot, the execution time of subsequent instructions may fluctuate (typically 50ms). In particular, when the robot continues to move from the paused state, the trigger position of subsequent non-motion instructions may change (not earlier than the original trigger point) as the maximum allowable transition area changes, depending on the size of the transition area given by the instruction. Therefore, use the motion I/O function for applications with high trigger position repeatability requirements (see the additional parameters for each motion instruction for how to use the motion I/O function).
- (8) In free transition mode, the transition length and path of the robot may change due to non-motion instructions. Assuming that the two motion displacements of the robot are S1 and S2, the actual transition length may vary depending on whether there are non-motion instructions between the two motion instructions.

① When there is no non-motion instruction between the two motion instructions, the robot begins to transition to the subsequent motion instruction when it enters the transition area. In this case, the theoretical transition time is consistent with the actual transition time, and the robot's transition length is not limited by the transition time.

② When there is a non-motion instruction between two motion instructions, since the robot begins to execute subsequent non-motion instructions when it enters the transition area, the execution time of the non-motion instructions will take up the time originally used for the transition of the motion instructions, which shortens the actual transition length of the robot due to time constraints.

Scope of instruction

- (1) Only valid for the main task
- (2) This feature is not supported in dynamic coordinate systems. The robot executes subsequent non-motion instructions after the motion has stopped.
- (3) This feature is not supported in fixed path transition mode. When NWait is not enabled in fixed path mode, the robot executes subsequent non-motion instructions after the motion has stopped. (See the additional parameters for each motion instruction for how to use the NWait parameter).
- (4) For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)

Example

```
SetFlyWait(On);  
Movj P[1],V[100],Z[CP],Tool[1];  
Set Out[4],On;  
SetFlyWait(Off);  
Movj P[2],V[100],Z[CP],Tool[1];  
Set Out[5],On;
```

The robot runs the above program from the P[0] position, and the steps are as follows:

- (1) Turn on the transition wait function through the instruction SetFlyWait;
- (2) The robot moves from the current point P[0] to P[1] and stays in that line, waiting until the motion ends;
- (3) After the robot stops moving, execute the instruction SetOut[4],On;
- (4) Turn off the transition wait function through the instruction SetFlyWait;
- (5) The robot moves from point P[1] to point P[2] and continuously monitors whether it enters the transition area;
- (6) After the robot enters the transition area (Z[CP] is half the instruction length), the robot keeps motion and starts to execute subsequent non-motion instruction Set Out[5]On simultaneously;

5.15SetAccuracyMode

Function: Sets the precision mode of the robot

Format: SetAccuracyMode(Normal/MidAccuracy);

Parameters:

0 - Normal (default mode):

When using the default mode, the robot adopts coarse positioning during continuous motion and fine positioning during discontinuous motion. The default mode allows high motion efficiency, but the accuracy during motion cannot be guaranteed. It is suitable for applications such as sorting and handling, which do not require high trajectory accuracy but require relatively short cycle time.

1 - MidAccuracy (medium accuracy mode)

Both continuous motion and motion with non-motion instructions use fine positioning. This mode offers shorter cycle time compared to Normal (default mode), but higher trajectory accuracy than Normal (default mode). When using medium accuracy mode, trajectory accuracy is only guaranteed by the control performance of the robot itself and the servo. This mode corresponds to the default mode for version 20 and earlier.

2 - HighAccuracy (high accuracy mode)

Both continuous motion and motion with non-motion instructions use fine positioning. The cycle time is consistent with MidAccuracy mode, which provides more than 30% improvement in trajectory accuracy compared to MidAccuracy mode. It is suitable for applications such as high-speed printing with higher trajectory accuracy requirements. This mode may cause the current to increase. In addition, the effect of using this mode at low speeds is not obvious. This mode is a new mode introduced in version 21.

3 - LowSpeedHighAccuracy (low speed high accuracy mode)

Both continuous motion and motion with non-motion instructions use fine positioning. The cycle is consistent with MidAccuracy mode, which improves the accuracy of the trajectory at low speeds and is suitable for cutting, welding, gluing and other applications where higher trajectory accuracy is required. Do not use this mode at high speeds, as this will cause vibration. This mode is not

supported at stage 1 of version 21 and the interface is reserved for the current version.

Note:

- (1) Accuracy settings only apply to free transition mode (default transition mode)
- (2) The differences between the modes are shown in the table below:

0 - Normal (default mode)	Accuracy	Lowest, accuracy cannot be guaranteed during continuous motion
	Efficiency	Highest
	Disadvantages	Accuracy cannot be guaranteed during continuous motion
	Applications	Applications with low accuracy requirements such as sorting and handling
1 - MidAccuracy (medium accuracy mode)	Accuracy	Average, higher than default mode
	Efficiency	Low
	Disadvantages	Low efficiency
	Applications	Applications with certain requirements for trajectory accuracy such as welding and gluing
2 - HighAccuracy (high accuracy mode)	Accuracy	High accuracy at high speeds
	Efficiency	Same with medium accuracy mode
	Disadvantages	Position stabilization time may be extended
	Applications	Applications with high requirements for trajectory accuracy such as high-speed printing
3 - LowSpeedHigh Accuracy (low speed high accuracy mode)	Accuracy	High accuracy at low speeds
	Efficiency	Same with medium accuracy mode
	Disadvantages	This mode cannot be applied at high speeds as it easily causes vibration
	Applications	Applications with high requirements for trajectory accuracy such as high-precision welding, gluing, etc.

Scope of instruction: Only effective in the main task.

Example:

```
Movj P[0],V[100],Z[0],Tool[1]; #Moves to preparation point
SetAccuracyMode(HighAccuracy);#Enters high accuracy mode
Movl P[1],V[100],Z[0],Tool[1];
Movl P[2],V[100],Z[0],Tool[1];
SetAccuracyMode(Normal);#Enters default mode
Movj P[0],V[100],Z[0],Tool[1]; #Moves to preparation point
```

5.16 GetTorque

Function: Read robot torque data

Format: Double GetTorque(int AxisNo);

Parameters:

AxisNo: Axis number, range 1 to 6

Scope of instruction: Only effective in the main task.

Return value: Robot torque data, double data

Example:

#Read the current data value for the first axis

D[6] = GetTorque(1);

5.17RapidMove

Function: Turns on or off the optimal trajectory function

Format: RapidMove(MotionType, Enable);

Parameters:

MotionType: Mode of the optimal trajectory, , which can be selected from CP/PTP/ALL

When MotionType=CP, Enable=ON, the CP instructions such as MovL/MovC/JumpL turn on optimal trajectory planning;

When MotionType=CP, Enable=OFF, the CP instructions such as MovL/MovC/JumpL turn off optimal trajectory planning;

When MotionType=PTP, Enable=ON, the PTP instructions such as MovJ/Jump turn on optimal trajectory planning;

When MotionType=PTP, Enable=OFF, the PTP instructions such as MovJ/Jump turn off optimal trajectory planning;

When MotionType=ALL, Enable=ON, all motion instructions turn on optimal trajectory planning;

When MotionType=ALL, Enable=OFF, all motion instructions turn off optimal trajectory planning;

Note: Currently, the optimal trajectory only supports PTP motion and is not valid for CP motion even if it is turned on.

Scope of instruction: Only effective in the main task. Motion instructions between RapidMove(*, ON) and RapidMove(*, OFF) turn on the optimal trajectory function.

Example:

Movj P[0],V[30],Z[0],Tool[1];

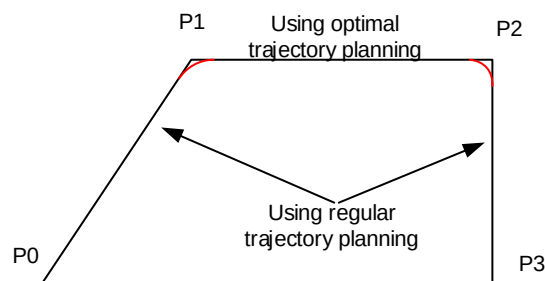
Movj P[1],V[30],Z[5],Tool[1];

RapidMove(PTP, ON);

Movj P[2],V[30],Z[5],Tool[1]; //This instruction uses optimal trajectory planning

RapidMove(PTP, OFF);

Movj P[3],V[30],Z[0],Tool[1];



Note: The load must be set before the optimal trajectory is turned on. If the user does not set the load or the load is less than 0.00001, the system believes that the user has forgotten to set the load. For safety, the system will automatically move according to the maximum load, and the movement will

be slowed down.

5.18 SetAcc

Function: Sets the acceleration of the motion segment

Description: This instruction is used to forcibly modify the acceleration parameters. After executing this instruction, the acceleration parameters in subsequent motion instructions are changed to the value of this instruction. Once set, this instruction remains in effect until SetAcc is executed again or the project is initialized. In general, the larger the SetAcc, the more likely it is to cause robot vibration. Please adjust the input parameters of the SetAcc instruction according to actual needs.

Format:

SetAcc(accValue,decValue);

SetAcc(OFF);

Parameters:

accValue: Forces the replacement of the acceleration parameter in the motion instruction, in percent, range 1 to 120. After the execution of SetAcc, the Acc parameter in Move, Jump and other instructions is replaced, with the SetAcc value as the acceleration percentage.

decValue: Forces the replacement of the deceleration parameter in the motion instruction, in percent, range 1 to 120. After the execution of SetAcc, the Dec parameter in Move, Jump and other instructions is replaced, with the SetAcc value as the deceleration percentage.

OFF: Cancels the acceleration setting of SetAcc(accValue, decValue) in Format 1. The motion instructions use their own Acc parameter.

Scope of instruction: Only effective in the main task.

Return value: /

Example:

```
Movj P[0],V[30],Z[0],Tool[1];
```

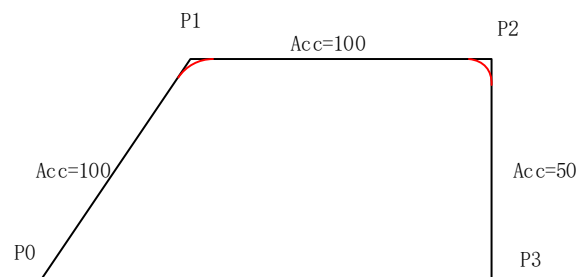
```
SetAcc (100,100);
```

```
Movj P[1],V[30],Z[5],Tool[1]; //The acceleration factor of this instruction is 100
```

```
Movj P[2],V[30],Z[5],Tool[1],Acc[50]; //The acceleration factor of this instruction is 100%
```

```
SetAcc (OFF);
```

```
Movj P[3],V[30],Z[0],Tool[1],Acc[50]; //The acceleration factor of this instruction is 50%
```



5.19 SLVSMODE

Function: Sets the self-learning vibration suppression mode

Description: This instruction is used to turn self-learning vibration suppression on or off

Format: SLVSMODE modePara;

Parameters:

modePara: Includes four modes: HighLevel, MidLevel, LowLevel, and Off;

Scope of instruction:

- 1) Only effective in main tasks, not supported in multitasking;
- 2) Project level: Initialization is triggered when the program is compiled and the program returns to the start of the main program, and this parameter restores the default value, which is Off;

Detailed description:

- 1) HighLevel, MidLevel and LowLevel indicate that the self-learning vibration suppression function is turned on, and Off indicates that the vibration suppression function is turned off;
- 2) The higher the level of vibration suppression, the better the effect of vibration suppression. In terms of vibration amplitude, $\text{HighLevel} \leq \text{MidLevel} \leq \text{LowLevel}$, and in terms of motion execution time, $\text{HighLevel} \geq \text{MidLevel} \geq \text{LowLevel}$;
- 3) When the self-learning vibration suppression function is required, it is recommended to use MidLevel first for testing. If the vibration does not meet the requirements, switch to HighLevel for testing. If the motion execution time does not meet the requirements, switch to LowLevel for testing;

Example:

- 1) In the following program, after the instruction SLVSMODE MidLevel is executed, the self-learning vibration suppression mode will be set to a medium-level mode;

For B[0]=0,B[0]<2,Step[1]

```
    Movj LP[0],V[30],Z[0],SLOn;  
    Delay T[1];  
    Movj LP[1],V[30],Z[0],SLOn;  
    Delay T[1];  
    SLVSMODE MidLevel;
```

EndFor;

5.20SLDataClear

Function: Clears learned data

Description: This instruction is used to clear data that has been learned. When it is called, the robot needs to perform self-learning again.

Format: SLDataClear ClearPara;

Parameters:

ClearPara: Includes All and Current, which respectively represent the erasure of all learning data and the erasure of learning data at the current position of the robot;

Scope of instruction: Only effective in main tasks, not supported in multitasking;

Detailed description:

- 1) When SLDataClear All is called, all learning data from previous learning will be restored to factory defaults.
- 2) After SLDataClear Current is called, the relevant learning data stored in the robot's current position will be cleared.
- 3) If the robot's self-learning effect is still poor after increasing the global motion speed (the robot's vibration is visible to the naked eyes), SLDataClear All can be called to clear all

historical learning data, and then self-learning can be re-started, or the robot can be moved to a position with poor self-learning effect and SLDataClear Current can be called to clear the learning data of the current position, and then re-start self-learning at that position.

- 4) If the weight or inertia of the end load of the robot changes significantly (more than 30%), SLDataClear All needs to be called to clear all historical learning data before learning again.
- 5) If you need to clear the historical learning data and re-learn, you generally only need to call the instruction SLDataClear All at the beginning of the program, and only need to call it once.
- 6) The instruction SLDataClear All is generally only used during debugging. After debugging is completed, it needs to be deleted or commented out. Otherwise, the learning data that has been learned from each SLDataClear All execution will be completely cleared, resulting in vibration suppression failure and the robot needs to learn again.

Example:

1) In the following program, after the instruction SLDataClear All is executed, all historical self-learning data will be cleared. When the motion instruction with SLOn or SLReset parameter is executed later, self-learning will be performed again, that is, self-learning will be performed upon the first execution of Movj LP[1], V[30], Z[0], SLON.

SLDataClear All;

For B[0]=0,B[0]<2,Step[1]

 Movj LP[0],V[30],Z[0];

 Delay T[1];

 Movj LP[1],V[30],Z[0],SLOn;

 Delay T[1];

 SLVSMODE MidLevel;

EndFor;

6 Process Control Instructions

6.1 #Comment

Function: Inputs comment

Description: Inserts comment in a program

Format: # Content

Parameters:

Content: Content of comment

Example: #Program is made at 10.

6.2 L-Goto

Function: Logical jump

Description: L is used to set program labels and is often used in conjunction with the jump instruction Goto to complete the jump action

Format:

L[labelNo]: #Cannot be repeated

.....

Goto L[labelNo];

Parameters:

labelNo: label number

Note: For loop instructions such as L-Goto, While-EndWhile, For-EndFor, etc., continuous execution of a large number of non-motion loops should be avoided as much as possible, such as executing assignment, reading PLC variables, etc. within the loop body without delay, which will cause too high controller CPU usage and slow response. The solution is to use Delay to increase the delay within the loop.

In addition: When using label instructions, it is not allowed to cross functions between L and GOTO L [], otherwise logical errors may occur.

Example:

START;

Movj P[0],V[30],Z[3],Tool[1];

L[1]: #Sets label 1

Movl P[1],V[30],Z[3],Tool[1];

Movl P[0],V[30],Z[3],Tool[1];

Goto L[1]; #Goes to label 1

END;

##Execution effect: First run to the position P[0], and then move back and forth between the two points P[1] and P[0].

6.3 Delay

Function: Delay

Description: Program delay, time controlled by parameter T

Format: Delay T;

Parameters:

T: Time, unit s, value range 0.000 to 65535.000. The time accuracy is typically 0.1s.

Example:

```
Delay T[5];      ##Delay by 5s
```

6.4 Include

Description: When it is needed to call functions of another module, use the instruction Include to declare that the module is referenced.

Format: Include "Module name.pro";

Parameters: Module name.

Example:

```
Include "Modell.pro";      Declares the reference to Modell.pro module
```

```
Start;
```

```
    Modell.fun1();
```

```
End;
```

Note: The instruction Include must be defined at the beginning of the program instruction, after the point data, and must be defined outside the function body.

6.5 Func

Function: Custom functions

Description: Functions are divided into main functions (entry functions for tasks) and custom functions. Each module can be composed of several functions, but each task has only one entry module and one entry function in that module; other modules cannot contain entry functions.

Format:

Format of main function

```
Start;
```

```
...function body...
```

```
End;
```

Custom unction format:

```
Func function name (parameter list)
```

```
    ...function body...
```

```
EndFunc
```

Parameter list:

The parameter list can be customized variables or arrays of bool, byte, int, float, double types. String variables or structs are not supported.

The parameters support pass in and pass out. The addition of sign & to the variable definition represents reference to the parameter, indicating that the variable supports pass out.

Example 1 (without parameter reference):

```
Func funAdd(int aa,int bb)
```

```
    R[1] = aa + bb;
```

```
EndFunc;
```

Example 2 (with parameter reference):

```
Func Add(int cc,int dd,&int Total)
    Total = cc + dd;
Endfunc;
```

Function call format:

1. Function call format for the current module: Function name (parameter list);
2. Function call format for other modules: Module name. Function name (parameter)##It is required to include the corresponding program name in the header of the program file

Example:

Main module ModelA.pro:

include "ModelB.pro";#If you need to call a function in ModelB.pro, you need to include the module

```
Func funAdd(int aa,int bb)
```

```
    R[1] = aa + bb;
```

```
EndFunc;
```

```
Start;
```

```
    #Calls functions in the same module
```

```
FunAdd(10,20);
```

```
#Calls functions in other modules
```

```
ModelB. funDec(30,10);
```

```
End;
```

Submodule ModelB.pro:

```
Func funDec(int aa,int bb)
```

```
    R[1] = aa - bb;
```

```
EndFunc;
```

6.6 Ret

Function: Returns from function

Description: Returns to the main function from the function, no longer executing the subsequent contents of the instruction in the function, but executing the statement following the main function.

Format: RET:

Note:

1. Often used with function calls;
2. If the starting line is set to execute from the main function:
 - (1) When the ret instruction is in the main function and the program executes to the line where ret is located, the running state will be set to end;
 - (2) When the ret instruction is in the function and the program executes to the line where ret is located, it will return from the function to the main function and no longer execute the instructions after the RET line in the function;
3. If the starting line is set to execute from the function, when the program executes to the line where

ret is located, the running state will be set to end;

6.7 If-Else-EndIf

Function: Conditional judgment

Description: Makes conditional judgment one by one, and executes the corresponding statement if the condition is met.

Format:

If condition1

 Stmnt1;

ElseIf condition2

 Stmnt2;

ElseIf condition3

 Stmnt3;

.....

Else

 Stmntn;

EndIf;

Parameters:

 Condition: Indicates condition

 Stmnt: Statement executed when the condition is met

Description:

The conditions can be a combination of multiple conditions.

The statement Stmnt can be several lines of instructions.

When If is used at the beginning, Else can be omitted. EndIf is indispensable as the end of the paragraph.

Example:

START;

If IN[1]==ON And IN[2]==ON

 Movj P[1],V[50],Z[3],Tool[1];

Else

 Movj P[2],V[50],Z[3],Tool[1];

 Movj P[3],V[50],Z[3],Tool[1];

EndIf;

End;

6.8 For-EndFor

Function: Loop statement with number of executions.

Description: First, an initialization assignment statement is executed, and then the condition is determined. If the condition is satisfied, the contents in the loop are executed. After the execution is

finished, one step is executed so that the variable in the initialization statement increments itself, and then the condition is determined. If the condition is satisfied, continue with the previous step until the condition does not hold.

Format:

```
For  InitExp, Condition, Step[n]
    Stmt;
EndFor;
```

Parameters:

InitExp: Initialization assignment statement, which is limited to assignment statement for B/R/LB/LR variables.

Condition: Indicates condition

n: Step, i.e., the increment of the corresponding variable each time after the content of the loop is executed.

Stmt: Statement executed when the condition is met.

Note: For loop instructions such as L-Goto, While-EndWhile, For-EndFor, etc., continuous execution of a large number of non-motion loops should be avoided as much as possible, such as executing assignment, reading PLC variables, etc. within the loop body without delay, which will cause too high controller CPU usage and slow response. The solution is to use Delay to increase the delay within the loop.

Example:

```
For B0=0,B0<5,Step[2]
    Movj P[2],V[50],Z[3],Tool[1];
    Movj P[3],V[50],Z[3],Tool[1];
EndFor;
```

Description: Two instructions Movj loop 3 times

6.9 Switch-Case-Default-EndSwitch

Function: Condition selection statement

Description: Selects a Case according to the variable after Switch and executes Stmt1 after Case. If no Case matches, the statement after Default will be executed.

Format:

Switch Var

Case value1:

 Stmt1;

 Break;

Case value2:

 Stmt2;

 Break;

.....

Default:

 Stmnt;

```
Break;  
EndSwitch;  
Parameters:
```

Var: Variables, which can be numerical variables of integer type and string variables.

Value: The value of the variable. Can be an integer, or a string corresponding to a string variable.

Stmt: Statement executed when the condition is met.

Extension:

Case A To B: Case A To B can be used to replace Case Value. Case A To B represents a selection within the range of Integer A to Integer B (A and B included). When a variable value is within this range, it indicates that this Case is matched.

Note:

In general, it is best to end each Case (including Default) paragraph with a break. If there is no Break at the end of a Case paragraph, continue to execute the next Case paragraph until the Break.

The Default statement can be omitted. If an entire conditional selection paragraph ends with Default, then Break must follow after the end of the Default contents. If an entire paragraph does not use Default, i.e. only Case is used, then Break must follow after the end of the contents of the last Case.

Example:

```
Switch B0
```

```
Case 1:
```

```
    Movj P[1],V[50],Z[3],Tool[1];
```

```
    Break;
```

```
Case 2 To 4:
```

```
    Movj P[1],V[50],Z[3],Tool[1];
```

```
    Movj P[2],V[50],Z[3],Tool[1];
```

```
    Break;
```

```
Case 5:
```

```
    Movj P[3],V[50],Z[3],Tool[1];
```

```
    Break;
```

```
Default:
```

```
    Movj P[4],V[50],Z[3],Tool[1];
```

```
    Break;
```

```
EndSwitch;
```

6.10 While-EndWhile

Function: Conditional loop

Description: Determine the condition, and if the condition is met, execute the statement within the loop. Once the condition is not satisfied, the program jumps out of the loop.

Format:

```
While Condition
```

```
    Stmt;
```

```
EndWhile;
```

Parameters:

Condition: Indicates condition

Stmt: Statement executed when the condition is met.

Description:

The conditions can be a combination of multiple conditions.

The statement Stmt can be several lines of instructions.

Note: For loop instructions such as L-Goto, While-EndWhile, For-EndFor, etc., continuous execution of a large number of non-motion loops should be avoided as much as possible, such as executing assignment, reading PLC variables, etc. within the loop body without delay, which will cause too high controller CPU usage and slow response. The solution is to use Delay to increase the delay within the loop.

Example:

START;

Movj P[1],V[50],Z[3],Tool[1];

While LB[0]<3

 Movj P[2],V[50],Z[3],Tool[1];

 Movj P[1],V[50],Z[3],Tool[1];

 Incr LB[0];

EndWhile;

End;

Description: The above run instruction loops three times and the whole program moves back and forth three times from P[1] to P[2].

6.11 Break

Function: Ends loop body

Description: Completely ends a loop, jumps out of the loop, and executes the statement after the loop. For example, jump out of the While or For loop, or jump out of a Switch statement after executing a Case segment. When multi-level loops are nested, only the loop at the current level is affected.

Format: Break;

Example:

START;

LB1 = 0;

For B1=0, B1<5, Step[1]

If LB1>1

 Break;

EndIf

Movj P[0],V[30],Z[0],Tool[1];

Movj P[1],V[30],Z[0],Tool[1];

Incr LB1;

EndFor;

END;
##Executes "Movj P[0], Movj P[1]" twice.

6.12 Continue

Function: Ends the current loop

Description: Skips the remaining statements in the current loop and executes the next loop. When multi-level loops are nested, only the loop at the current level is affected.

Format: Continue

Example:

```
START;  
LB1 = 0;  
For B1=0, B1<5, Step[1]  
If LB1==1  
    Continue;  
EndIf  
Movj P[0],V[30],Z[0],Tool[1];  
Movj P[1],V[30],Z[0],Tool[1];  
Incr LB1;  
EndFor;  
END;  
##Executes "Movj P[0], Movj P[1]" four times
```

6.13 WaitInPos

Description: Waits until the motion reaches the specified proportion or the encoder position is reached

Format: WaitInPos(Value/Fine/Corner)

Parameters:

(1) Value: Percentage of current motion displacement (optional)

- ① Parameter type: Integer
- ② Value range 1 to 100, default value 100
- ③ Note: /

(2) Fine: Wait for current motion until encoder position is arrived (optional)

- ① Parameter type: Keyword
- ② Note: The default parameter indicates that the accuracy is Fine.

(3) Corner: Transition midpoint (optional)

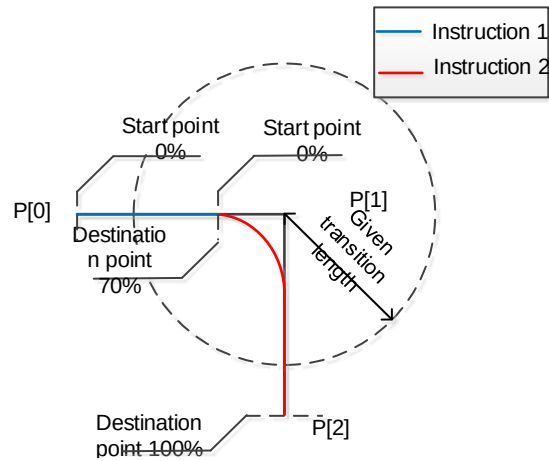
- ① Parameter type: Keyword
- ② Note: Near the transition midpoint of the previous motion and the current motion.

Note:

- (1) This instruction is only supported for use in motion tasks.
- (2) This instruction is only supported in static coordinate systems. In dynamic coordinate

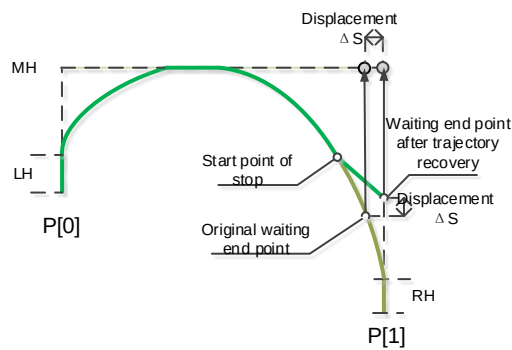
systems, the robot will wait for the motion to stop before executing subsequent instructions.

- (3) When this instruction is used in Jump, JumpL, or motion with a transition path, the accuracy of the displacement value may not be guaranteed. Please adjust it according to the actual situation.
- (4) Because the motion path of the transition area cannot be guaranteed, there may be deviations when automatically calculating the transition midpoint using Corner parameters. Please adjust it according to the actual situation.
- (5) Displacement percentage D is referenced to the previous motion. When the target points of multiple consecutive motion instructions are too close or repeated, only the first motion instruction can produce motion. In this case, the robot takes the first motion instruction as reference and executes when the robot reaches the specified displacement percentage.
- (6) When the bot is currently in a position greater than or equal to the specified displacement percentage, the robot no longer waits and executes subsequent instructions directly. In particular, when a stop signal is received, the robot will no longer detect the current displacement percentage, so subsequent instructions will not be triggered until the robot resumes operation from the stop state.
- (7) The default accuracy when the WaitInPos instruction does not contain input parameters is Fine, indicating that the encoder is arrived. WaitInPos (100) has an accuracy of Z [0], indicating that the planning is in place. Select a different level of accuracy based on the actual accuracy requirements.
- (8) Note that when the last two motion instructions form a transition, the starting point of the transition is generally the end of the previous motion instruction and the starting point of the next motion instruction. As shown in Figs. 2-3, assuming that the previous and next instructions are instruction 1 and instruction 2, the transition to instruction 2 begins when instruction 1 reaches 70%. At this point, it can be considered that instruction 1 ends running at the transition starting point and instruction 2 starts to run.



Figs. 2-3 Schematic diagram of displacement percentage with transition

- (9) When using Corner as a parameter of the instruction WaitInPos, if there is no transition between the current motion and an earlier motion, the instruction WaitInPos ends waiting when the current motion starts to execute and the robot immediately executes subsequent instructions.
- (10) During the Jump motion, the robot calculates the approximate displacement according to the projection of the current point in the vertical and horizontal directions. Therefore, when the motion path of the robot changes, the position corresponding to the same displacement will also change. If the robot reaches the displacement value corresponding to the original end wait when the Jump command is instruction during motion and restarted, the robot will likely end the wait at a point diagonally above the original end wait.



Scope of instruction (including usage scenario limits, whether it is effective in multitasking)

- (1) Only valid for the main task
- (2) This feature is not supported under dynamic following.
- (3) SetFlyWait settings are reset to default values when one of the following conditions occurs. Subject to reset conditions of project level variables.

Example:

Example 1:

MovJ P[1], V[100], Z[CP], NWAIT, Tool[1];

WaitInPos(30);

If In[5] == On;

Set Out[4], On;

EndIf;

WaitInPos(80);

Set Out[5], On;

MovJ P[2], V[100], Z[0], Tool[1];

Resolution:

After the motion reaches 30%, determine In[5]==On and execute Set Out[4], On

Executes Set Out[5], On after the motion reaches 80%.

Maintain the transition when passing P[1] point, with the theoretical transition area less than 20%.

Example 2:

MovJ P[1], V[100], Z[CP], Tool[1];

WaitInPos(30);

B0 = B0 + 1;

WaitInPos(80);

Set Out[5], On;

MovJ P[2], V[100], Z[0], Tool[1];

Resolution:

Execute B0 = B0 + 1 when the motion reaches 50%

Execute Set Out[5], On when the motion reaches 80%.

Maintain the transition when passing P[1] point, with the theoretical transition area less than 20%.

Example 3:

MovJ P[1], V[100], Z[0], Tool[1];

WaitInPos(30);

Set Out[4],On;

WaitInPos(100);

Set Out[5],On;

WaitInPos;

Set Out[6],On;

MovJ P[2],V[100],Z[0],Tool[1];

Resolution:

Execute Set Out[4], Set Out[5]On when the motion reaches 100%

Execute Set Out[6],On when the motion reaches 100% and the encoder reaches the arrival accuracy range and stays for the set time

Stop briefly when reaching P[1] point, no transition.

Example 4:

MovJ P[1], V[100], Z[CP] ,Tool[1];

MovJ P[2],V[100],Z[0],NWAIT,Tool[1];

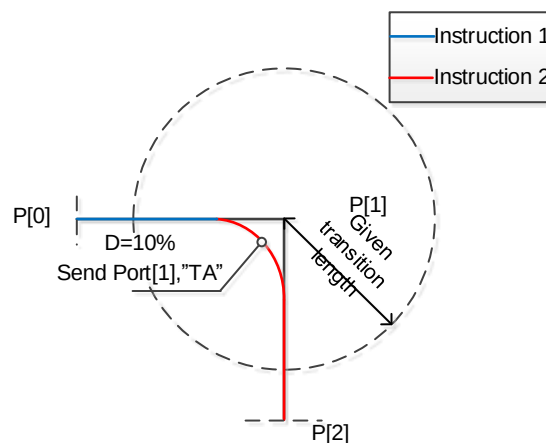
WaitInPos(10)##WaitInPos(Corner) can also be used

Send Port[1], "TA";

Resolution:

The robot starts transitioning before reaching P [1] and executes Send Port [1], "TA", after MovJ P [2] reaches 10% motion.

Then stop at P[2].



7 Task Control Instructions

7.1 Xqt

Function: Starts task

Description: Use this instruction to start a master task or an instruction-based PLC task, which allows multitasking programs to run together.

Format: Xqt Index, filepath;

Parameters:

Index: Task number, currently only 3.

filepath: Program path name associated with an instruction-based task, such as "ABC.pro".

Example:

Xqt 3, "ABC.pro";

7.2 Halt

Function: Halts task

Description: Pauses selected task

Format: Halt Index;

Parameters:

Index: Task number. Current can only be 3.

Example:

Halt 3;

7.3 Resume

Function: Resumes task

Description: Resumes the selected task

Format: Resume Index;

Parameters:

Index: Task number. Current can only be 3.

Example:

Resume 3;

7.4 Quit

Function: Stops and quits the task

Description: Stops and quits selected task

Format: Quit Index;

Parameters:

Index: Task number. Current can only be 3.

Example:
Quit 3;

7.5 Pause

Function: Stop

Description: Stops operation. Equivalent to the function of the Stop button on the teach pendant. To resume the operation, press the Run key. This instruction is often used to view variables during commissioning.

Format: Pause;

Note: 1. The instruction Pause in the static task stops the main task and the dynamic task;

8 Operation Instructions

8.1 Variable Definition

Function: Defines the variables

Format:

Individual variable definition: Variable type variable name;

Array variable definition: Variable type variable name[number of elements] [number of elements]... ;

Array data initialization: Variable type variable name [number of elements] = {element1, element2...};

BOOL variable name [number of elements][number of elements] = {{element1, element2...}, {element1, element2...}};

BOOL variable name [number of elements][number of elements][number of elements]= {{element1, element2...},{element1, element2...},{element1, element2...},{element1, element2...}};

The initial value for the items missing during initialization is 0.

Parameters:

Variable type: Supports the definition of custom variables for struct types such as String, Bool, Byte, Int, Float, Double, etc.

Variable name: The variable name is composed of letters, numbers, and underscores and can only start with letters. The name length cannot exceed 32 characters.

Number of array elements: Maximum support for triple array, total array length cannot exceed 10000.

Note:

1. When defining variables, they can be initialized. But when initializing a string variable, there is a limit on the length of the string, which cannot exceed 100 characters.
2. The value of the string variable supports Chinese characters. It should be noted that the string operations in the instruction are all in single-byte operations. Chinese encoding method is not the same as English, occupying 2 bytes width, so you need to consider the difference between the two when doing string operations. For example, if one character is taken from a Chinese string, the printed character will be garbled.

3.

Example:

Int aaa; ##Defines variables

Float ccc[100][10]; ##Defines array

Float ddd[2][2] = {{1,2},{3,4}}; ##Initializes array, note: If the number of initialized expressions is less than the number of dimensions of the array, the value not covered is 0 by default.

8.2 Numerical Operations

8.2.1 Incr

Function: Self-increment of numeric variables

Format:

Incr var;

Parameters:

var: Integer variable, supports B/R/LB/LR/Int/Byte type variables

Example:

B[1]=1;

Incr B[1]; ##B[1] increased by 1, and the value changes to 2

8.2.2 Decr

Function: Self-decrement of numeric variables

Format:

Decr var;

Parameters:

var: Integer variable, supports B/R/LB/LR/Int/Byte type variables

Example:

B[1]=1;

Decr B[1]; ##B[1] decreased by 1, and the value changes to 0

8.2.3 Sin

Function: Sinusoidal operation

Description: Sinusoidal operation.

Format: <Var>= Sin(<Exp>);

Parameters:

<Exp>: Operative expression, in angles

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]=Sin(60+D[2]);

8.2.4 Cos

Function: Cosine operation

Description: Cosine calculation, in angles

Format: <Var>= Cos(<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]=Cos(60+D[2]);

8.2.5 Tan

Function: Tangent operation

Description: Tangent operation, in angles

Format: <Var>= Tan(<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Tan (60+D[2]);

8.2.6 Asin

Function: Asin operation

Description: Asine operation.

Format: <Var>= Asin(<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double type, in angles

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Asin (60+D[2]);

8.2.7 Acos

Function: Acos operation

Description: Acos operation.

Format: <Var>= Acos(<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double type, in angles

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Acos(60+D[2]);

8.2.8 Atan

Function: Atan operation

Description: Atan operation.

Format: <Var>= Atan (<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double type, in angles

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Atan (60+D[2]);

8.2.9 Sqrt

Function: Sqrt operation

Description: Sqrt operation.

Format: <Var>= Sqrt (<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Sqrt (9); ##D[1]=3

8.2.10Pow

Function: The y-th power of x

Description: Calculates the y-th power of x.

Format: <Var>= Pow(<Exp_x>, <Exp_y>);

Parameters:

<Exp_x>:Expression for base

<Exp_y>:Expression for power

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Pow(2,3); ##D[1]=8

8.2.11Abs

Function: Absolute value

Description: Takes the absolute value.

Format: <Var>= Abs (<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= Abs (-2); ##D[1]=2

8.2.12Max

Function: Comparison of maximum values

Description: Takes the maximum of two values.

Format: <Var>= Max (<Exp_x>, <Exp_y>);

Parameters:

<Exp_x>, <Exp_y>: Expression for comparison

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

B[1]=1;

R[1]=2;

D[1]= Max(B[1],R[1]); ##D[1]=2

8.2.13Min

Function: Minimum value comparison

Description: Takes the minimum of two values.

Format: <Var>= Min (<Exp_x>, <Exp_y>);

Parameters:

<Exp_x>, <Exp_y>: Expression for comparison

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

B[1]=1;

R[1]=2;

D[1]= Min(B[1],R[1]); ##D[1]=1

8.2.14AngleToRad

Function: Angle to radian

Description: Converts angle to radian.

Format: <Var>= AngleToRad (<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= AngleToRad (90);

8.2.15RadToAngle

Function: Radian to angle

Description: Converts radian to angle.

Format: <Var>= RadToAngle (<Exp>);

Parameters:

<Exp>:Expression

Return value:

Return value type: double

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will take a forced transformation of the data.

Example:

D[1]= RadToAngle (3.14);

8.3 String Operations

8.3.1 Left

Description: Takes a few characters on the left side of a string, as shown in the format below, n characters on the left side are taken.

Format: StrDest= Left(StrSource, n);

Parameters:

StrSource: Source string object

n: Number of characters (English)

Return value: StrDest: Truncated string

Example

Str1 = "abcdefg";

Str2 = Left(Str1,2) ##Takes two left characters of Str1, and the result is Str2="ab"

8.3.2 Right

Description: Takes a few characters on the right side of a string, as shown in the format below, n characters on the right side are taken.

Format: StrDest= Right(StrSource, n);

Parameter:

StrSource: Source string object

n: The number of characters on the right side to be taken

Return value: StrDest: Truncated string

Description: Number of n characters (English)

Example

Str1 = "abcdefg";

Str2 = Right(Str1,2) ##Takes two right characters of Str1 and the result is Str2 = "fg"

8.3.3 Mid

Function: Takes the middle of a string

Description: Takes the middle characters of the string, as shown in the following format, n characters starting from i are taken

Format: StrDest = Mid(StrSource,i,n);

Parameters:

StrSource: Source string object

i: Start index

n: The number of characters to be taken (English)

Return value:

StrDest: Truncated string

Example

```
Str1 = "abcdefg";
```

```
Str2 = Mid(Str1,3,3)    ##Takes three characters starting from the third character in Str1, and the  
result is Str2="def"
```

8.3.4 GetAt

Function: Gets a character in a string

Description: Gets a certain character in the string, as shown in the format below, the character with index i is taken

Format: StrDest = GetAt(StrSource, i);

Parameters:

StrSource: Source string object

i: Index of character to be taken

Return value: StrDest: Truncated string

Example

```
Str1 = "abcdefg";
```

```
Str2 = GetAt(Str1, 3)    ##Gets the character indexed with 3 in Str1, and the result is Str2 = "d"
```

8.3.5 StrFindEnd

Function: Reverse lookup

Description: Finds the last occurrence of the specified character in the string. Return the index of the character if find succeeds, or -1 if failed.

Format: Index = StrFindEnd (StrSource ,StrItem);

Parameters:

StrSource: Source string object

StrItem: Object to find in the source string

Return value: Index: The index that was queried. Failure returns -1.

Example

```
LB1=StrFindEnd ("abcaba", "a"); ##Result is LB1=5
```

8.3.6 StrRePlace

Function: String replacement

Description: Replaces old string with new string in the source string

Format: StrDest = StrRePlace (StrSource,OldSubStr, NewSubStr);

Parameters:

StrSource: Source string

OldSubStr: Old substring to be replaced

NewSubStr: New substring for replacement

Return value: StrDest: Replaced string

Example

```
String Str1="aoe";
```

```
Str1= StrRePlace (Str1,"a","b");  ##Result is Str1 = "boe"
```

8.3.7 StrReverse

Function: Reverses string

Description: Reverses order of characters in string

Format: StrDest = StrReverse(StrSource);

Parameters: StrSource: Source string

Return value: StrDest: Reversed string

Example

```
String Str1="aoe";
```

```
Str1= StrReverse(Str1); ##Result is Str1 = "eo"
```

8.3.8 Strlen

Function: Takes the length of the string

Description: Takes the length of the string

Format: length = Strlen(StrSource);

Parameters: StrSource: Source string

Return value: length: String length

Example

```
Str1 = "abcd";
```

```
LB[1] = Strlen(Str1);          ##Result is LB1 = 4
```

8.3.9 StrFind

Function: String lookup

Description: Finds the index of a specified substring in a string

Format: Index = StrFind(StrSource,SubStr);

Parameters:

StrSource: Source string

SubStr: Substring to find

Return value: Index: Specifies the starting index of the substring. Returns -1 when not found.

Example

```
Str1 = "abcde";
```

```
LR[1] = Strlen(Str1,"cd");      ##Result is LR1 = 2
```

8.3.10 TrimLeft

Function: String left trim

Description: Removes the spaces to the left of the string

Format: StrDest = TrimLeft(StrSource);

Parameters: StrSource: Source string

Return value: StrDest: Trimmed string

Example

```
Str1 = "  abc";
```

```
Str1 = TrimLeft(Str1);
```

```
##Result is Str1 = "abc"
```

8.3.11TrimRight

Function: String right trim

Description: Removes the spaces to the right of the string

Format: StrDest = TrimRight(StrSource);

Parameters: StrSource: Source string

Return value: StrDest: Trimmed string

Example

```
Str1 = "abc  ";
```

```
Str1 = TrimRight (Str1);
```

```
##Result is Str1 = "abc"
```

8.3.12Strcmp

Function: String comparison

Description: Compares the size of two strings. The two strings are compared character by character (by ASCII size) from left to right until a different character appears or until '\0' is encountered.

Return the difference between the ASCII codes of the first two different characters, or 0 if they are all the same.

Format: value = Strcmp (Str1, Str2);

Parameters:

Str1: String being compared

Str2: String being compared

Return value: value: Result of comparison. The value is the difference between the ASCII codes of the first two different characters, or 0 if they are all the same.

Example

```
Str1="ABxC";
```

```
Str2="ABzH";
```

```
R1=Strcmp(Str1,Str2)  ##R1=-2
```

8.3.13Printf

Function: Formats string

Description: Formats the string in the specified form, same as the C language sprintf(char *buffer,

const char *format, [argument] ...)

Format:

Sprintf(StrValue, format, argument);

Parameters:

StrValue: String to be formatted

Format: Format string. Can be a combination of: %[flag][[width](#)][.precision]type

Flag:

-: Indicates left-aligned output; if omitted, indicates right-aligned output.

0: Indicates the specified blank space is filled with 0; if omitted, indicates the specified blank space is not filled in.

[width](#):

Total length after formatting. If the length of the data is less than [width](#), a space is added to the left end. If it is greater than [width](#), it is output based on the actual number of digits.

precision:

Specifies the number of decimal places. The default number of decimal places is 6 when not specified.

Type:

d Signed decimal integer

f Floating point number (inclus String

Parameters: (Optional parameter) A list of variables that can be any type of data. Depending on the Format parameter, the function expects a series of additional parameters, each containing a value to replace the format descriptor in the format string.

Example:

String Str1;

R[1]=20;

D[1]=12 .123

Sprintf(Str1,"R[1]=%d And D[1]=%.2f",R[1],D[1]); ##Result is Str1 = "R[1]=20 And D[1]=12.12!"

8.4 Numeric Conversion

8.4.1 Caps

Function: Converts string to uppercase

Description: Converts string to uppercase

Format: StrDest = Caps(StrSource);

Parameters:

Return value:

StrDest: Converted string

Note: Caps does not process non-English letters.

Example:

Str1 = "aB12";

```
Str2 = Caps (Str1);    ##Str2 = "AB12"
```

8.4.2 LowCase

Function: Converts string to lowercase

Description: Converts string to lowercase

Format: StrDest =LowCase(StrSource);

Parameters:

StrSource: Source string

Return value:

StrDest: Converted string

Note: LowCase does not process non-English letters.

Example:

```
Str1 = "aB12";
```

```
Str3 = LowCase(Str1);  ##Str 3= "ab12"
```

8.4.3 RToStr

Function: Converts integer variables to string variables

Description: Converts integer variables to string variables

Format: StrValue = RToStr(Var)

Parameters:

Var: Integer variable to be converted

Example:

```
R[1] = 45;
```

```
Str4 = RToStr(R[1]);    ##Str4 = "45"
```

8.4.4 DToStr

Function: Converts double variables to string variables

Description: Converts double variables to string variables

Format: StrValue = DToStr(Var,m,n);

Parameters:

Var: Double variable to be converted

m: The number of integers after conversion

n: The number of decimal places after conversion

Return value:

StrValue: Converted string variable

Note:

m defines the number of integers after conversion.

- If $m \leq$ the actual number of integers, the integer part is not processed and all will be output;
- If $m >$ the actual number of integers, then fill in the number of integers with spaces on the

left.

n defines the number of decimal places after conversion.

- If $n \leq$ the actual number of decimal places, the decimal portion is rounded to an approximation;
- If $n >$ the actual number of decimal places, then complete the string with zeros from the right.

Example:

LD[1] = 1.36;

Str5 = DToStr(LD[1],2,1); ##Result is Str5 = "1.4", note that there is a space before 1

8.4.5 StrToR

Function: Converts strings to integer data

Description: Converts strings to integer data

Format: Var = StrToR(StrSource);

Parameters:

StrSource: Source string to be converted.

String format:

- Strings starting with 0x indicate hexadecimal strings;
- Strings starting with 0d (or none) indicate decimal strings;
- Strings starting with 0b indicate binary strings.

Return value:

Var: Converted integer variables

Description:

- 1) StrToR recognizes only the preceding numeric bits (left to right) and stops when it encounters a non-numeric bit; if the first bit is a non-numeric bit, it returns 0.

Format description:

- Decimal string: Consists of the symbols "+", "-", "0-9", excluding decimal points
 - Hexadecimal string: Consists of numbers "0-9" and letters "A-F", "a-f", excluding decimal points and symbols "+", "-"
 - Binary string: Consists of numbers "0" and "1", excluding decimal points and symbols "+", "-"
- 2) When using hexadecimal strings and binary strings, please note that the strings must conform to the format of hexadecimal strings and binary strings, and must not contain symbols such as "-" and ".", otherwise the converted value will be incorrect.
 - 3) The hexadecimal numbers support conversion of up to 64bit data, that is, 8-bit hexadecimal numbers.
 - 4) Binary numbers support conversion of up to 32bit data, that is, 32-bit binary numbers.

Example:

Str1 = "a12";

Str2 = "12a3";

Str3 = "1234"

Str4 = "0x0A";

Str5= "0xFFFF";

```
Str6= "0b10101100";
```

```
R[1] = StrToR(Str1);      ##R[1]= 0
R[2] = StrToR(Str2);      ##R[2] = 12
R[3] = StrToR(Str3);      ##R[3] = 1234
R[4] = StrToR(Str4);      ##R[4] = 10
R[5] = StrToR(Str5);      ##R[5] = 65535
R[6] = StrToR(Str6);      ##R[6] = 172
```

8.4.6 StrToD

Function: Converts string to double precision data

Description: Converts string to double precision data

Format: Var = StrToD(StrSource);

Parameters:

StrSource: Format of source string to be converted.

- Strings starting with 0x indicate hexadecimal strings;
- Strings starting with 0d (or none) indicate decimal strings;
- Strings starting with 0b indicate binary strings.

Description:

- 1) StrToD recognizes only the preceding numeric bits (left to right) and stops when it encounters a non-numeric bit; if the first bit is a non-numeric bit, it returns 0.

Format description:

- Decimal string: Consists of symbols "+", "-", and numbers "0-9", containing decimal points
 - Hexadecimal string: Consists of numbers "0-9" and letters "A-F", "a-f\\"", excluding decimal points and symbols "+", "-"
 - Binary string: Consists of numbers "0" and "1", excluding decimal points and symbols "+", "-"
- 2) When using hexadecimal strings and binary strings, please note that the strings must conform to the format of hexadecimal strings and binary strings. For example, the string cannot contain symbols such as "-" and ".", otherwise the converted value will be incorrect
 - 3) The hexadecimal numbers support conversion of up to 64bit data, that is, 8-bit hexadecimal numbers
 - 4) Binary numbers support conversion of up to 32bit data, that is, 32-bit binary numbers

Example:

```
Str1 = "123.456"
```

```
Str2= "0x0A"
```

```
Str3 = "0b10101100";
```

```
LD[1] = StrToD(Str1);      ##LD[1] = 123.456
```

```
LD[2] = StrToD(Str2);      ##LD[2]= 10
```

```
LD[3] = StrToD(Str3);      ##LD[3] = 172
```

8.4.7 PToStr

Function: Converts point data to a string

Description: Converts point data to a string

Format:

```
StrValue = PToStr(P);
```

Parameters:

P: Point to be converted

Return value:

StrValue: Converted string

Note: The first six coordinate values retain three decimal places after conversion.

Example:

```
P[0]=(10,0,0,0,0,0),(1,0,0,0),(1,0,0);
```

```
Str[1]=PToStr(P[0]);
```

```
##Str[1] result is 10.000,0.000,0.000,0.000,0.000,0.000;1,0,0,0; 1,0,0;
```

8.4.8 BToAscii

Function: Converts Byte/B/LB type variables to corresponding characters by ASCII code

Description: Converts Byte/B/LB type variables to corresponding characters by ASCII code

Format:

```
StrValue = BToAscii(Var);
```

Parameters:

Var: A Byte/B/LB type variable. If the input variable is a non-Byte type variable, it will be forced to convert to a Byte type variable first before the statement is executed.

Return value:

StrValue: Converted string.

Example:

```
String NewStr;
```

```
LB[1]=65;
```

```
NewStr = BToAscii(LB[1]); ##Result is NewStr="A"
```

8.4.9 HexToStr

Function: Converts hexadecimal string

Description: Converts the string StrSource represented by a hexadecimal number into the characters corresponding to that hexadecimal number, and stores the string composed of the characters in StrDest.

Format:

```
length = HexToStr(StrSource,StrDest);
```

Parameters:

StrSource: Incoming parameter, i.e. string to be converted

StrDest: Outgoing parameter, i.e. variable stored in converted string

Return value:

length: Length of the converted string

Example:

Define a string variable StrSource = "213A716A3477"

↓

Call the instruction HexToStr to convert StrSource to StrDest = "!:qj4w"

8.4.10 StrToHex

Function: Converts a string to a hexadecimal string

Description: Converts data stored in a string to a hexadecimal string.

Format:

length = StrToHex(StrSource,StrDest);

Parameters:

StrSource: Incoming parameter, i.e. string to be converted

StrDest: Outgoing parameter, i.e. variable stored in converted string

Return value:

length: Length of converted string

Example:

GetPort receives data and stores data in string StrSource

! : q j 4 w

↓

Call StrToHex to convert it to a hexadecimal string and stored it in StrDest

21 3A 71 6A 34 77

8.4.11 GetStrAscii

Function: Converts string to ASCII code

Description: Converts string to ASCII code and store the code in an array of B variables

Format: RetNum = GetStrAscii(StrSource,num,B);

Parameters:

StrSource: Source string

num: The number of characters to convert to ASCII code. Range: 0-255 and cannot greater than the length of the string.

B: B/LB variable, where data is stored in a series of consecutive B variables starting from that B variable.

Return value:

RetNum: The number of characters successfully converted

Note:

Ensure that the number of B/LB variables is not exceeded.

Example:

String str1 = "abc";

LR[1] = GetStrAscii (str1,3,LB[1]); ##LB[1], LB[2], and LB[3] are 97, 98, 99, respectively,

LR[1]=3

8.4.12 StrGetData

Function: Gets string data

Description: Gets data from a string, stores it in the specified variable, and returns the number of data retrieved

Format: num = StrGetData(StrSource, Separator, Data);

Parameters:

StrSource: Source string. Currently, only local strings and Port are supported, and Port represents the string in the receive buffer.

Separator: Separator, which is a string

Data: Data objects stored after segmentation, in the following three forms.

- B/R/D/LB/LR/LD[***]: Data is stored in an array of current B/R/D/LB/LR/LD variables.
- P/LP[***]: Data is stored in the format of the P variable P/LP[***].
- PR/LPR[***]: Data is stored in the format of the PR variable PR/LPR[***].
- Custom array variables, such as int aa[3]; StrGetData(StrSource, Separator, aa[0]).
Note that when the number of split variables exceeds the range of the array, automatic truncation occurs.

Return value:

num: The number of data retrieved

Example 1:

Str1 = "123And45Andggg"

B[0]= StrGetData(Str1,"And",R[0]);

Description: The string is divided into three strings: "123", "45", and "ggg", resulting in B [0]=3, R [0]=123, R [1]=45, and R [2]=0.

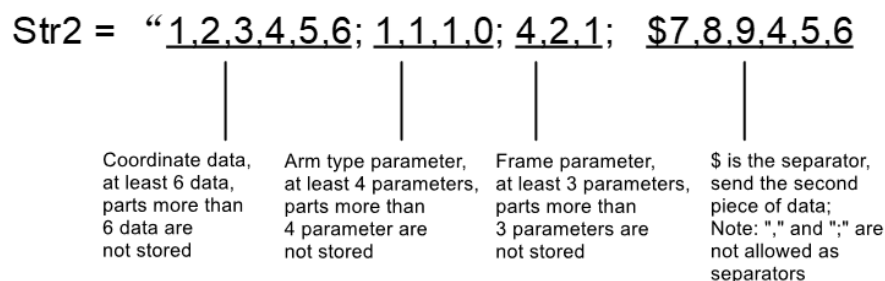
Example 2:

Str2 = "1,2,3,4,5,6;1,1,1,0;4,2,1;\$7,8,9,4,5,6;"

B[0]= StrGetData(Str2,"\$",P[1]);

Description:

1) Description of Str2:



2) The first separated string "1,2,3,4,5,6;1,1,1,0;4,2,1;" is stored in P[1], B[0]=1, as shown in the

following figure:

Name	J1/X	J2/Y	J3/Z	J4/A	J5/B	J6/C	Coord	Tool	User
P[000]	0.000	0.000	5.079	0.000	0.000	0.000	1	0	0

The four ArmTypes are 1, 1, 1, and 0, respectively.

3) For the second separated string "7,8,9,4,5,6", the number of Arm type parameter and the coordinate parameters do not meet the specification, thus will not be stored in P[2], B[1]=0.

8.4.13 CVDataToPose

Function: Parses vision data into position variables

Description: Extracts data from string StrSource sent from vision equipment into position variable P

Format 1: CVDataToPose(StrSource, P, PVarMemset);

Parameters:

StrSource: Incoming parameter, a source string object sent from vision equipment:

Format: CVXY, X, Y, X, Y..., the frame head is CVXY, and the data will be parsed in the format of [X, Y]

Format: CVXY, X, Y, X, Y..., the frame head is CVXY, and the data will be parsed in the format of [X, Y]

P: Outgoing parameter, P variable used to store point data

PVarMemset: Initializes the outgoing P variable

Return value: Number of converted position variables, failed: <=0.

Description:

1. The data extracted from StrSource only contains XY or XYA and is placed in the corresponding position in P; the remaining values in P (such as arm parameters, tool numbers, etc.) are filled by PVarMemset correspondingly.

Example:

```
P[0] = {(0,0,0,0,0,0),(1,0,0,0),(2,0,0)};
```

```
String StrSource = "CVXY,10,20,11,21";
```

```
R[0] = CVDataToPose(StrSource,P[1],P[0]);
```

```
Print R[0];
```

```
Print P[1];
```

```
Print P[2];
```

Result:

```
2
```

```
10,20,0,0,0,0;1,0,0,0;2,0,0;
```

```
11,21,0,0,0,0;1,0,0,0;2,0,0;
```

Description: Extracts data from string StrSource sent from vision equipment into position variable P. The string may include the type (L) of visually recognized object. When the string includes L, not only the XYA data is saved to P, but also the L data is saved to DVar; when the string does not include L, only the XYA data is saved to P, and DVar is not used.

Format 2: CVDataToPose(StrSource, P, DVar,PVarMemset);

Parameters:

StrSource: Incoming parameter, a source string object sent from vision equipment:

Format: CVXY, X, Y, X, Y..., the frame head is CVXY, and the data will be parsed in the format of [X, Y]

Format: CVXY, X, Y, X, Y..., the frame head is CVXY, and the data will be parsed in the format of [X, Y]

Format: CVXYAL,X,Y,A,L,X,Y,A,L...,the frame header is CVXYAL, the data will be parsed according to [X,Y,A,L] format, and the L data will be saved to the starting DVar variable array

P: Outgoing parameter, P variable used to store point data

DVar: Stores L parameter

PVarMemset: Initializes the outgoing P variable

Return value: Number of converted position variables

Description:

1. The data extracted from StrSource only contains XY or XYA and is placed in the corresponding position in P; the remaining values in P (such as arm parameters, tool numbers, etc.) are filled by PVarMemset correspondingly.
2. L parameter representation

Example:

```
P[0] = {(0,0,50,0,0,0),(1,0,0,0),(2,0,0)};  
String StrSource = "CVXYAL,10,20,30,3,11,21,31,4";  
R[0] = CVDDataToPose(StrSource,P[1],D[0],P[0]);  
Print R[0];  
Print P[1];  
Print P[2];  
Print D[0];  
Print D[1];
```

Result:

```
2  
10,20,50,30,0,0;1,0,0,0;2,0,0;  
11,21,50,31,0,0;1,0,0,0;2,0,0;  
3  
4
```

8.5 Coordinate Operations

8.5.1 Cnvrt

Function: Point conversion

Description: Performs coordinate conversion of points

Format:

```
Cnvrt (FromP, ToP,type);  
Cnvrt (FromP, ToP,type, TrigP);
```

Parameters:

FromP: Point to be converted

ToP: Converted point

Type: Coordinate system type, which determines which coordinate system the FromP is converted to. There are four types:

Joint Joint coordinate system. The point will be converted to a joint coordinate system point with the tool number and user number unchanged.

World World coordinate system. The point will be converted to a world coordinate system point (equivalent to taking a point in the world coordinate system, with the coordinate value being the pose value of the flange end of the robot body in the world coordinate system) with the tool number and user number unchanged.

Tool[i] Tool coordinate system. The point will be converted to a tool coordinate system point (equivalent to taking a point in the tool coordinate system, with the coordinate value being the pose value of TCP in the world coordinate system) with the user number unchanged.

User[i] User coordinate system. The point will be converted to a user coordinate system point (equivalent to taking a point in the user coordinate system, with the coordinate value being the pose value of TCP in the user coordinate system) with the tool number unchanged.

TrigP: (Optional parameter) The shooting point of the dynamic camera, which is valid only when the coordinate system number of FromP is 6, and errors are reported in all other cases.

Note:

When this instruction is used, the coordinate system number of P [i] must be 1 to 6 and cannot be 7.

When the coordinate system number of P[i] is set to 5, it will be possible to convert points in the fixed camera coordinate system to the robot's joint/world/tool/user coordinate system.

When the coordinate system number of P[i] is set to 6, it will be possible to convert points in the dynamic camera coordinate system to the robot's joint/world/tool/user coordinate system.

Example

```
Cnvt (P[1],P[2], Joint);        #Coonverts P[1] into the joint coordinate system and assigns a value to P[2]
```

8.5.2 Dist

Function: Find the linear distance between two points

Description: Find the linear distance between two points

Format: Var = Dist(FromP, ToP);

Parameters:

FromP: Start point

ToP: Destination point

Return value:

Return value type: double

Var: Numeric variable to store the returned value

Note: Use the coordinate system of the first position variable as a reference to determine the linear distance of the two points.

8.5.3 GetCurPoint

Function: Gets coordinate values under the current joint or world coordinate system

Description: Gets coordinate values under the current joint or world coordinate system

Format 1: GetCurPoint(CoordType,Index,Var);

Format 2: GetCurPoint(CoordType, Index,P);

Parameters:

CoordType: Only 1 or 2 can be taken.

- 1 Indicates the joint coordinate system, taking the current joint value (J1,J2,J3,J4,J5,J6)
- 2 Indicates the world coordinate system, taking the current world coordinate value (X,Y,Z,A,B,C). (Tools excluded)

Index: Valid only in format 1 and is the subscript index of the coordinate value. Indicates taking one of the 6 coordinate values.

Var: Numeric variable, which stores a coordinate value from the acquired position variable

P: Saves position variable

Note:

If the last parameter is Var, the data is stored in a numerical variable. In this case, the subscript is taken from 0 to 5, indicating that one of the six coordinate values is taken for saving.

If the last parameter is P, the data is stored in the P variable. In this case, the subscript can be any of 0 to 5 and is meaningless.

Example

GetCurPoint(2,1,D[1]) #Takes the Y coordinate value of the current point under the world coordinate system and assigns a value to D[1]

GetCurPoint(1,0,P[1]) #Takes the value of the current point in the joint coordinate system and assigns a value to P[1]

8.5.4 P[i]=

Function: Assigns a value to a point

Description: Assigns a value to a position variable

Format:

P=(X,Y,Z,A,B,C),(ArmType[0],ArmType[1],ArmType[2],ArmType[3]),(CoordNo,ToolNo,UserNo);

or

P={X,Y,Z,A,B,C),(ArmType[0],ArmType[1],ArmType[2],ArmType[3]),(CoordNo,ToolNo,UserNo)};

Parameters:

(X,Y,Z,A,B,C): Joint value (J1, J2, J3, J4, J5, J6) when coordinate system number is 1, pose value(X, Y, Z, A, B, C) when coordinate system number is 2, 3 or 4.

(ArmType[0],ArmType[1],ArmType[2],ArmType[3]): Robot arm parameters. Set arm parameters depending on the robot model.

CoordNo: Coordinate system number

ToolNo: Tool number

UserNo: User number

Example:

P[3] = (10,50,30,40,50,60),(1,-1,1,0),(4,1,1);

Note: On the teach pendant, you can use the **Original_P** button to get the original value of the point in the file.

8.5.5 OffsetJ

Function: Joint offset

Description: Joint offset

Format:

OffsetJ(P[***],PR[***]);

OffsetJ(P[***],J1[*],J2[*]...);

Parameters:

P [***]: Position variable subject to offset

PR[***]: Offset variable

J1[*],J2[*]...: Select one or more, up to six of them. J1[*]: Move in J1 direction of the target point by a specified angle.

Note:

- When the instruction OffsetJ is used, the six coordinate values of PR are the joint rotation angles. The joint value will be automatically calculated based on the position variable P for offset.
- When used in conjunction with motion instructions, such as Movj OffsetJ(P, PR), if the motion instructions contain parameters such as Tool, User, etc., these parameters are invalid for OffsetJ.

Example 1:

PR[1]=(10,20,0,0,0,0);

Movj OffsetJ(P[1],PR[1]) ,V[30], Z[0],Tool[1];

On the basis of P[1], the robot rotates the first joint by an additional 10° and the second joint by an additional 20°.

Example 2:

PR[1]=(10,20,0,0,0,0);

P[2]=OffsetJ(P[1],PR[1]);

Movj P[2],V[30],Z[0],Tool[1];

Example 3:

Movj OffsetJ(P[1],J1[10],J2[20]) ,V[30], Z[0],Tool[1];

The above three examples have the same effect.

8.5.6 OffsetT

Function: Offset along the current tool position

Description: Offset along the current tool position

Format:

OffsetT(P[***],PR[***]);

OffsetT(P[***],X[*],Y[*]...);

Parameters:

P [***]: Position variable subject to offset

PR[***]: Offset variable

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*]: Indicates offset in the X-direction of the tool coordinate system of the target point by the specified amount of displacement.

Features:

- Use OffsetT to offset on the basis of the current tool coordinate system. PR is movement and rotation (X, Y, Z, A, B, C). Offset sequence: X>Y>Z>A>B>C
- If used in conjunction with a motion instruction, such as MovJ OffsetT(P, PR)..., the motion instruction must include a Tool parameter and no User parameter.

Example:

PR[1]=(10,0,0,10,20,30);

MovJ OffsetT(P[1],PR[1]) ,V[30], Z[0],Tool[1];

The position of the target point to be reached. That is, the position to be reached by moving 10 mm in the X direction, rotating 10° around Z axis, then 20° around Y axis, and finally 30° around X axis in the current tool coordinate system and based on P[1].

PR[1]=(10,0,0,10,20,30);

P[2]=OffsetT(P[1],PR[1]);

MovJ P[2],V[30],Z[0],Tool[1];

The destination position can also be reached.

MovJ OffsetT(P[1],X[10]) ,V[30], Z[0],Tool[1];

Indicates that the end point of the tool is offset by 10 mm under the coordinate system of tool 1 on the basis of P[1].

8.5.7 Offset

Function: Offset in Cartesian frame

Description: Offset of a target point under a Cartesian reference frame. The target point may be the TCP or flange center point. The reference frame may be a user or world coordinate system.

Format:

Offset (P[***],PR[***]);

Offset(P[***],X[*],Y[*]...);

Parameters:

P [***]: Position variable subject to offset

PR[***]: Offset variable

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*]: Indicates the offset in the X direction of the base coordinate system of the target point by the specified distance.

Description:

PR is movement and rotation (X, Y, Z, A, B, C). Offset sequence: X>Y>Z>A>B>C

- For P = Offset (P, PR), directly perform offset on coordinate values of position variables. A target offset point and a reference coordinate system are designated by a coordinate system number because the meanings of coordinate values are designated by a coordinate system.
 - When the coordinate system number in the position variable is 1, the center point of the flange offsets in

base coordinate system.

- When the coordinate system number in the position variable is 2, the center point of the flange offsets in base coordinate system.
- When the coordinate system number in the position variable is 3, the TCP offsets in the base coordinate system. The tool is specified by the tool number of the position variable.
- When the coordinate system number in the position variable is 4, the TCP offsets under the user coordinate system, the tool is specified by the tool number of the position variable, and the user coordinate system is specified by the user number of the position variable.

You can use the instruction Cnvr to change the position variable to be represented under the specified coordinate system, and then with P=Offset(P,PR), you can specify the target point and reference system of the translation. We recommend this for clarity.

Example 1:

Take points in the base coordinate system, and the flange moves in the base coordinate system.

```
P[1]={ (300,100,-30,0,0,0)(1,0,0,0(2,1,0));
```

```
PR[1]=(10,0,0,0,0,0);
```

```
P[2]=Offset(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0];
```

Since P [1] is a point taken in the base coordinate system (coordinate system number 2), it is the translation of the flange center relative to the base coordinate. Note that the target point is not TCP at this point.

If a translation of the TCP relative to the base coordinate system is required, refer to the following usage:

```
P[1]={ (300,100,-30,0,0,0),(1,0,0,0),(2,1,0));
```

```
Cnvr(P[1]P[1]Tool[1]);##Converts P[1] to be expressed under the tool coordinate system
```

```
P[2]=Offset(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

If P[1] is a point taken in the tool coordinate system (coordinate system number 3), it can be converted without the instruction Cnvr.

If due to certain special needs, the TCP end needs to be translated in the specified user coordinate system User[1], refer to the following usage:

```
P[1]={ (300,100,-30,0,0,0),(1,0,0,0),(2,1,0));
```

```
Cnvr(P[1]P[1]User[1]);##Converts P[1] to be expressed in User1
```

```
P[2]=Offset(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

- Movj Offset (P, PR) refers to the translation of the target point in the specified coordinate system.

The parameters Tool and User in the motion instruction is described as follows for the target point and the coordinate system to be translated.

	With	Without
Tool	The target point indicates end of Tcp	The target point indicates end of robot
User	The translation coordinate system is	The translation coordinate system is

	the user coordinate system	the base coordinate system
--	----------------------------	----------------------------

Example 2:

Example of tool translation in the base coordinate system

P[1]={ (300,100,-30,0,0,0),(1,0,0,0),(2,1,0)};

Movj Offset(P[1],PR1),V[30],Z[0], Tool[1];

At this point, as long as the motion instruction Movj includes the Tool parameter but not the User parameter, the tool translates in the base coordinate system. The position variable P can be taken in any coordinate system.

Example of changing tools first and then translating

P[1]={ (300,100,-30,0,0,0),(1,0,0,0),(2,1,0)};

Movj Offset(P[1],PR1),V[30],Z[0],Tool[2];

Note:

1. When Mov does not include the Tool and User parameters, it is the translation with the target point as the end of the robot flange, otherwise it is the translation with the target point as the end of TCP.
 2. When Mov does not include the User parameter, it is the translation of the target point in the base coordinate system, otherwise it is the translation in the user coordinate system.
- To change to the user coordinate system, the condition "P's coordinate system number is 4" must be met. The reference point must be switched from a point in the original user coordinate system to a point in the user coordinate system, and then translated in the direction of the user coordinate system. Otherwise, it means direct translation in the User coordinate system in the instruction.

Example 3:

When switching to the user coordinate system first and then translating, it is important to note that switching to the user coordinate system must meet the conditions.

When P is not a point taken in the user coordinate system, the starting point of the translation does not change because the condition of switching to the user coordinate system is not met, but the translation follows the User coordinate system in the instruction.

When P is a point taken in the user coordinate system, first switch to the user coordinate system (the starting point of the translation changes) and then translate along the user coordinate system.

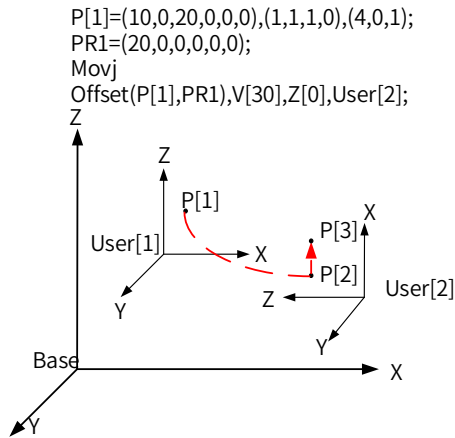


Figure 1

Figure 1:
Scenario: Movj has the User parameter and the coordinate system number of P is 4.
Feature: The absolute position of the positional variable changes before translation occurs. Finally, position P[3] is reached.

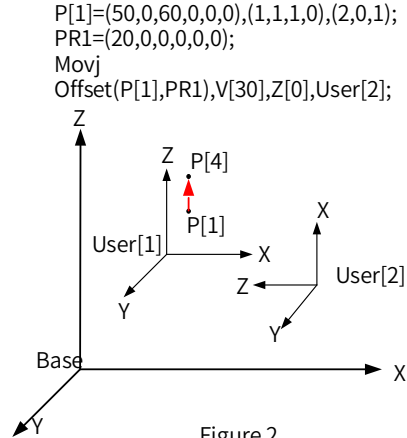


Figure 2

Figure 2:
Scenario: Movj has the User parameter and the coordinate system number of P is 1 or 2.
Feature: The absolute position of the positional variable does not Change and translation occurs directly. Finally, position P[4] is reached.

8.5.8 Msft

Function: Calculates translation

Description: Calculates translation variable between two position variables

Format 1:

$PR = \text{Msft}(\text{FromP}, \text{ToP});$

Parameters:

FromP: Start Point

ToP: End point

Return value:

PR:PR/LPR variable, as the calculation result

Format 2:

$PR = \text{Msft}(\text{FromP}, \text{ToP}, \text{Tcp});$

Parameters:

FromP: Start point

ToP: End point

Tcp: Translation of the end of the tool in the tool coordinate system

Return value:

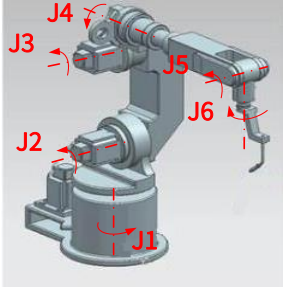
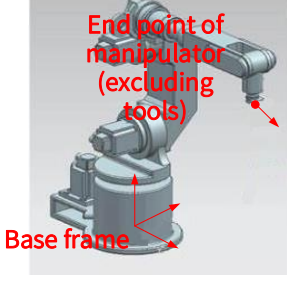
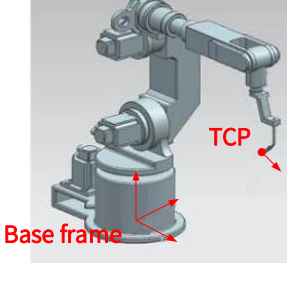
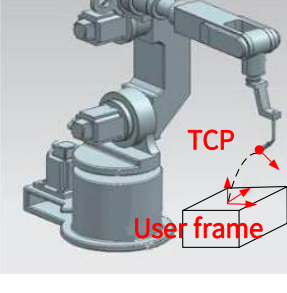
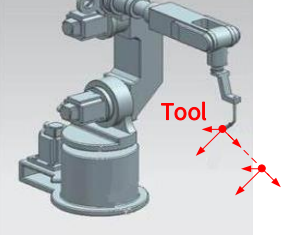
PR:PR/LPR variable, as the calculation result

Description:

Format 1: Calculates the translation variable from FromP to ToP. The translation variable is measured according to the coordinate system used by P[1].

Format 2: Calculates the translation variable from FromP to ToP. The translation variable is measured according to the current pose of P[1].

See the table below for details:

Joint translation calculation	Conditions of use: The coordinate system number of P[1] is 1. Typical usage form: $P[1] = (0,0,0,0,90,0),(0,0,0,1)(1,0,0);$ $P[2] = (10,10,10,10,100,10),(0,0,0,1)(1,0,0);$ $PR[1] = \text{Msft}(P[1],P[2]);$ Note: When P[2] is a non-joint point, it is first converted to a point in the joint coordinate system.	
Translation of robot body end along the base coordinate system	Conditions of use: The coordinate system number of P[1] is 2. Typical usage form: $P[1] = (390,-330,329,1,-8,-179),(0,0,0,1)(2,0,0);$ $P[2] = (390,-320,325,1,-8,-179),(0,0,0,1)(2,0,0);$ $PR[1] = \text{Msft}(P[1],P[2]);$ Note: When P[2] is a non-base coordinate point, it is first converted to a point at the end of the robot in the base coordinate system before the translation calculations.	
TCP translation along the base coordinate system	Typical usage form: The coordinate system number of P[1] is 3. $P[1] = (390,-330,329,1,-8,-179),(0,0,0,1)(3,1,0);$ $P[2] = (390,-320,325,1,-8,-179),(0,0,0,1)(3,1,0);$ $PR[1] = \text{Msft}(P[1],P[2]);$ Note: When P[2] is a non-tool point, it is first converted to a point at the end of the tool in the base coordinate system before the translation calculations.	
TCP translation along the user coordinate system	Typical usage form: The coordinate system number of P[1] is 4. $P[1] = (20,0,20,0,0,0),(0,0,0,1)(4,1,2);$ $P[2] = (30,0,10,0,0,0),(0,0,0,1)(4,1,2);$ $PR[1] = \text{Msft}(P[1],P[2]);$ Note: When P[2] is a non-user coordinate point, it is first converted to a point at the end of the tool in the user coordinate system before the translation calculations.	
TCP translation along the tool coordinate system	Typical usage form: Msft instruction with Tcp parameter $P[1] = (390,-330,329,1,-8,-179),(0,0,0,1)(3,1,0);$ $P[2] = (390,-320,325,1,-8,-179),(0,0,0,1)(3,1,0);$ $PR[1] = \text{Msft}(P[1],P[2],\text{Tcp});$ Note: Move from the current tool position P[1] to P[2] without changing the pose.	

--	--	--

Note: Translation variables are not allowed to be acquired when the arm parameters are inconsistent (in 6-joint robots, translation variables are not allowed to be calculated when the ArmType[3] parameters are inconsistent; in SCARA robots, translation variables are not allowed to be calculated when the ArmType[0-2] parameters are inconsistent).

8.5.9 EOffsOn

Function: Turns on path offset

Description: Turns on the offset of overall path. The final path of the motion will undergo additional offset along the X, Y and Z directions in the base coordinate system. Once turned on, it takes effect until the instruction EOffsOff turns off the path offset.

Format: EOffsOn(X,Y,Z);

Parameters:

X: Offset in the X direction in the base coordinate system

Y: Offset in the Y direction in the base coordinate system

Z: Offset in the Z direction in the base coordinate system

Example:

```
Movj P[1],V[30],Z[0],Tool[1];
```

```
EOffsOn(10,0,0);
```

```
Movl P[2],V[30],Z[0],Tool[1];
```

```
Movl P[3],V[30],Z[0],Tool[1];
```

```
EOffsOff;
```

Note:

1. EOffsOn must be used in pairs with EOffsOff, and EOffsOn must be used before EOffsOff can be used.
2. Path offset only acts on the instructions between EOffsOn and EOffsOff. The path offset is independent of the program logic and is only context dependent, as follows:

```
Movj P[1],V[30],Z[0],Tool[1];
```

```
B[1] = 2;
```

```
EOffsOn(10,0,0); ##Regardless of whether the instruction is executed or not, the offset will be turned on since here
```

```
Movl P[2],V[30],Z[0],Tool[1];##Offset
```

```
If B[1]>1
```

```
    Goto L[1];
```

```
EndIf;
```

```
L[2]:
```

```
Movl P[3],V[30],Z[0],Tool[1]; ##Here, regardless of from where the instruction jumps, it is subject to offset because it is between EOffsOn and EOffSetoff
```

```
Func1();##Here is a function call, no offset for motion instruction in the function
```

```
EOffsOff;
```

```
L[1]:
```

```
Movl P[4],V[30],Z[0],Tool[1];##This instruction is outside of EOffsOn and EOffSetoff, so even if it jumps here, there is no offset here
```

Goto L[2];

3. When the coordinate system number of P variable is 7, offset is not turned on. (This instruction is not included in the palletizing robot) MovToPut, MovToGet, MovFromPut, MovFromGet commands do not turn on offset.
4. Although the motion instruction in which the instruction EOffsOn takes effect has nothing to do with logic, the value of the PR variable in the instruction EOffsOn takes effect at the time of execution. Therefore, do not add conditional judgment before the instruction EOffsOn to prevent the offset variable of the instruction EOffsOn from taking effect, for example:

If B[1] > 1

 EOffsOn(10,0,0);

EndIf;

Movl P[2],V[30],Z[0],Tool[1];

Movl P[3],V[30],Z[0],Tool[1];

EOffsOff;

In the above program, the offset is effective, but the offset variable corresponding to the offset is not effective, that is, the value of the PR variable is the previous value (0,0,0).

8.5.10EOffsOff

Function: Turns off path offset

Description: Turns off the offset of path. When EOffsOn is used, path offset is always turned on until it is turned off by EOffsOff.

Format: EOffsOff;

For details, see [EOffsOn](#).

8.5.11P=Offset

Function: Point offset

Description: Point-based translation

Format 1:

DestP = OffSetJ(SourceP,PR);

DestP = OffSetJ(SourceP,J1[*],J2[*]...);

Parameter:

SourceP: Source point

PR:PR/LPR variable, as the translation amount

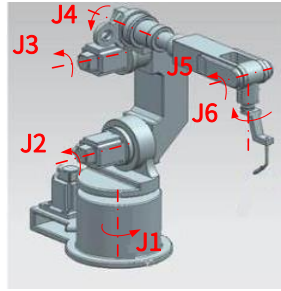
J1[*],J2[*],J3[*]...: Select one or more, up to six of them. J1[*]: Indicates that the target point is translated by a specified angle in the X direction in the joint coordinate system.

Return value:

DestP: Point after offset

Description of format 1:

Joint translation It refers to the offset in the joint coordinate system, and the value of PR is the joint offset angle (J1, J2, J3, J4, J5, J6).



Example:

```
PR[1]=(10,20,0,0,0,0);
```

```
P[2]=OffsetJ(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

##On the basis of P[1], the robot rotates the first joint by an additional 10 °and the second joint by an additional 20 °.

or

```
P[2]=OffsetJ(P[1],J1[10],J2[20]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

Format 2:

```
DestP = OffSetT(SourceP,PR);
```

```
DestP = OffSetT(SourceP,X[*],Y[*]...);
```

Parameters:

SourceP: Source point

PR/LPR variable, as the translation amount

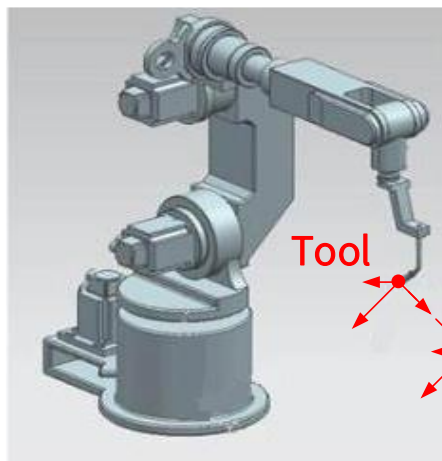
X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*]: Indicates translation in the X-direction of the tool coordinate system of the target point by the specified amount of displacement.

Return value:

DestP: Point after offset

Description of format 2:

Translation on the basis of the current tool coordinate system. PR is movement and rotation (X, Y, Z, A, B, C). Translation sequence: X>Y>Z>A>B>C



Example:

```
PR[1]=(10,0,30,0,0,0);
```

```
P[2]=OffsetT(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

or

```
P[2]=OffsetT(P[1],X[10],Z[30]);
```

```
Movj P[2],V[30],Z[0],Tool[1];
```

The position of the target point to be reached is a position after moving 10 mm along the X direction and 30 mm along the Z direction in the current tool coordinate system based on P[1].

Format 3:

```
DestP = Offset(SourceP,PR);
```

```
DestP = Offset(SourceP,X[*],Y[*],...);
```

Parameters:

SourceP: Source point

PR: PR/LPR variable, as the translation amount

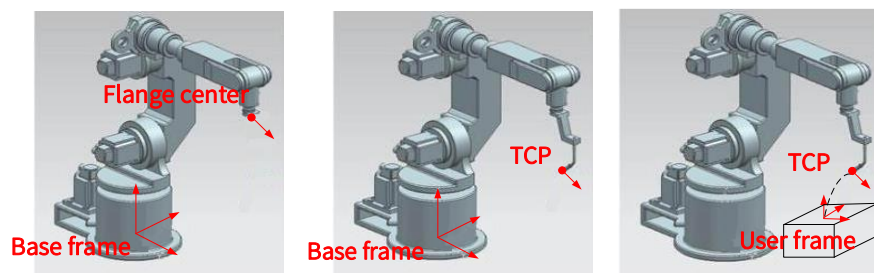
X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*]: Indicates that the target point is translated by a specified angle in the X direction in the joint coordinate system.

Return value:

DestP: Point after offset

Description of format 3:

Offset indicates translation of a target point under a Cartesian reference system. The target point may be the TCP or flange center point. The reference system may be a user or base coordinate system. PR is movement and rotation (X, Y, Z, A, B, C). Translation sequence: X>Y>Z>A>B>C
Several offset translations



A translated target point and a reference coordinate system are designated by a coordinate system number because the meanings of coordinate values are designated by a coordinate system.

- ✧ When the coordinate system number in the position variable is 1, the center point of the flange translates in base coordinate system.
- ✧ When the coordinate system number in the position variable is 2, the center point of the flange translates in base coordinate system.
- ✧ When the coordinate system number in the position variable is 3, the TCP translates in the base coordinate system. The tool is specified by the tool number of the position variable.
- ✧ When the coordinate system number in the position variable is 4, the TCP translates under the user coordinate system, the tool is specified by the tool number of the position variable, and the user

coordinate system is specified by the user number of the position variable.

Recommendation: You can use the instruction Cnvrt to change the position variable to be represented under the specified coordinate system, and then with P=Offset(P,PR), you can specify the target point and reference system of the translation.

Example:

Take points in the base coordinate system, and the flange moves in the base coordinate system.

```
P[1] = (3390,-330,329,1,-8,-179;0,0,1;2,1,0) ;##Also works when the coordinate system number is 1
```

```
PR1=(10,0,0,0,0,0);
```

```
P[2]=Offset(P[1],PR1);
```

```
Movj P[2],V[30],Z[0];
```

or

```
P[1] = (3390,-330,329,1,-8,-179;0,0,1;2,1,0) ;#Also works when the coordinate system number is 1
```

```
P[2]=Offset(P[1],X[10]);
```

```
Movj P[2],V[30],Z[0];
```

Since P [1] is a point taken in the base coordinate system (coordinate system number 2), it is the translation of the flange center relative to the base coordinate. Note that the target point is not TCP at this point.

If the translation of the TCP relative base coordinate system is required, refer to the follows:

```
P[1] = (390,-330,329,1,-8,-179;0,0,0,1;2,1,0);
```

```
Cnvrt(P[1]P[1]Tool[1]);##Converts P[1] to be expressed under the tool coordinate system
```

```
P[2]=Offset(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0];
```

If P[1] is a point taken in the tool coordinate system (coordinate system number 3), it can be converted without the instruction Cnvrt.

For example, if due to certain special needs, the TCP end needs to be translated relative to an additional user coordinate system User[1], refer to the following usage:

```
P[1] = (390,-330,329,1,-8,-179;0,0,0,1;2,1,0);
```

```
Cnvrt(P[1]P[1]User[1]);##Converts P[1] to be expressed in User1
```

```
P[2]=Offset(P[1],PR[1]);
```

```
Movj P[2],V[30],Z[0];
```

8.5.12PR Sum

Function: PR sum/difference

Description: Addition and subtraction of two translation variables

Format 1: PR[3] = PR[1] + PR[2];

Format 2: PR[3] = PR[1] – PR[2];

Description: Directly perform numerical addition and subtraction operations on two translation variables and assign values to another translation variable. It also applies to the local translation variable LPR.

Example: $PR[3] = PR[1] - PR[2];$

8.5.13 PR[i]=

Function: PR assignment

Description: Direct assignment to translation variable

Format: $PR = (X,Y,Z,A,B,C);$

or $PR = (J1,J2,J3,J4,J5,J6);$

Parameters: (X,Y,Z,A,B,C): Parameter value as the translation amount.

(J1,J2,J3,J4,J5,J6):Parameter value as the joint translation amount.

Return value: PR: PR/LPR variable, assignment object.

Note: PR variable can represent translation variable in Cartesian coordinate system or joint coordinate system. For access to single element of translation variable: $PR[*]$. X/Y/Z/A/B/C represent the first to sixth elements of $PR[*]$, which can represent the value of translation variable element in Cartesian coordinate system or the value of joint translation variable.

Example 1:

$PR[1] = (110,120,130,10,50,60);$ #Direct assignment

Example 2:

$LB[1] = 110;$

$LR[1] = 120;$

$LD[1] = 130;$

$B[1] = 10;$

$R[1] = 50;$

$D[1] = 60;$

$LPR[1] = (LB[1], LR[1], LD[1], B[1], R[1], D[1]);$ ##Use variable for indirect assignment

8.5.14 LoadPointFromFile

Function: Loads point file

Description: Loads points into the system from point file

Format: LoadPointFromFile(PtFile, Var);

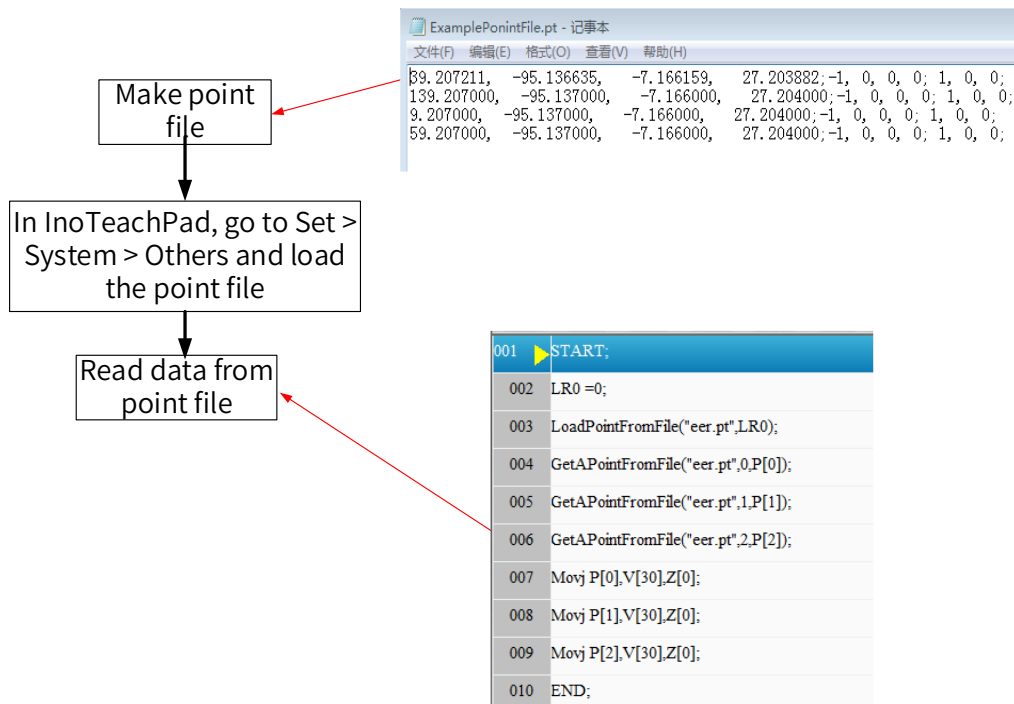
Parameters:

PtFile: Point file name

Var: Integer numeric variable that returns the total number of position variables in the point file

Description: Often used in conjunction with GetAPointFromFile to get location variables from external point files. A common process for using point files is as follows:

Use of point file



Case: Load the point information from the point file eer.pt, and assign the first three points to P[0], P[1], and P[2].

Note:

If no position variables are pre-defined in the program (e.g., P[0], P[1], P[2] in the above program), the instruction GetAPointFromFile will report an error.

Description:

Point files are suffixed with ".pt". The file contents are data information of position variables. One line indicates one piece of position variable information. For the format of every line, refer to the definition of "position variable". Each line is divided into 3 small paragraphs: the first six parameters for the first paragraph, coordinate system value; the middle four parameters for the second paragraph, arm parameters; the last three parameters for the third paragraph, the coordinate system number, tool number, user number, respectively. Paragraphs are separated from each other by ";", and the numbers within the paragraphs are separated by "," and terminated by ";".

You can specify 1 to 6 coordinate values, and replace the missing values with 0 if there are less than 6 values.

Format requirements:

Coordinate values, arm parameters, and coordinate system parameters are separated by a ";".

The data in the coordinate values, arm parameters, and coordinate system parameters are separated by ",".

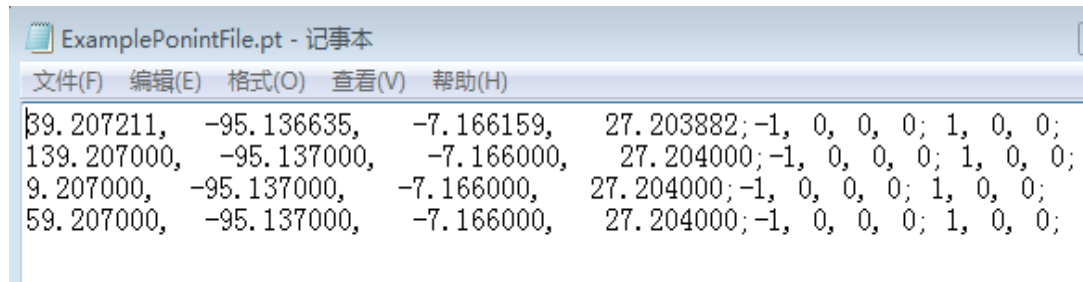
This instruction parses the required data in order (coordinate values; arm parameters; coordinate system parameters), and any excess data is not parsed.

Note: This command does not have strict requirements on the format of the string, and the string can be parsed as long as it is filled in according to the specification. When the data format does not

conform to the common format, no error is reported. Users need to ensure the correct format of the point data when using this instruction.

In most cases, the point data input by the instruction is not complete. In order for the user to enter a part of the parameters first through the instruction, the instruction does not check the legality of the coordinate values, and the legality of the data can only be checked when running or calculating.

An example is shown below:



8.5.15GetAPointFromFile

Function: Gets points from point file

Description: Loads a single point into the position variable of the program from the system

Format: GetAPointFromFile(PtFile, Index , P);

Parameters:

PtFile: Point file path name

Index: Index number (line number) of the point in the file

P: Extracted points are placed in this position variable

Description: See the description of LoadPointFromFile

Note:

If no position variables are pre-defined in the program, the instruction GetAPointFromFile will cause an error.

8.5.16WriteAPointToFile

Function: PtFile: Point file path name

Description: Specifies a line in the point file to write the points to.

Format: WriteAPointToFile(PtFile, Index, P);

Parameters:

PtFile: Point file path name

Index: Index number (line number) of the point to be saved in the file

P: Position variable to be stored in the point file

Description: The instruction SavePointFile must be called after writing the point file to save it

Example:

```
WriteAPointToFile("eer.pt", 0, P[0]);
```

```
WriteAPointToFile("eer.pt", 1, P[1]);
```

```
SavePointFile ("eer.pt");
```

8.5.17SavePointFile

Function: Saves point file

Description: Saves point file

Format: SavePointFile (PtFile);

Parameters: PtFile: Point file name

8.5.18Lpallet

(1) Three-point method

Name LPallet

Function LPallet definition

Description Sets the local pallet variable. Often used in conjunction with P=Pallet for using simple pallets.

The three points $P[i], P[j], P[k]$ form a parallelogram, called the "boundary point" of the pallet. $P[i]$ specifies the first point on the pallet, the line between $P[j]$ and $P[i]$ specifies the row direction of the pallet, and the line between $P[k]$ and $P[i]$ specifies the column direction of the pallet. The principle of this instruction is to create a pallet boundary based on the three input points, and set the pallet model based on the number of rows, columns, layers, and layer height. In the future, only the information of rows, columns, and layers needs to be used to move to the specified position on the pallet.

Format LPallet LPalletNo, $P[i], P[j], P[k], nRow, nColumn, nLay, LayHeight$;

Parameter LPalletNo: The serial number of the local pallet variable, and the value range is 0 to 255

$P[i], P[j], P[k]$: Defines three points of the pallet

nRow: Number of rows, value range 1 to 1000

nColumn: Number of columns, value range 1 to 1000

nLay: Number of layers, value range 1 to 1000

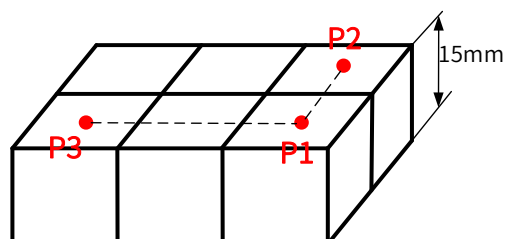
LayHeight: Layer height, value range 0 to 2000, in mm

Note:

1. You can define pallet as a single row or column, but you cannot define a pallet as a point.
2. When the number of pallet layers is greater than 1, any two points are not allowed to be duplicated, otherwise the calculation of pallet points will fail.

Example:

LPallet 1, $P[1], P[2], P[3], 2, 3, 1, 15$;



(2) Four-point method

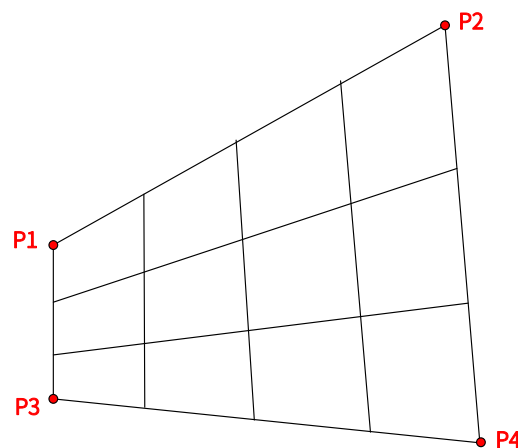
Name	LPallet
Function	4-point definition of LPallet
Description	Sets the local pallet variable by 4 points. Often used in conjunction with P=Pallet for using simple pallets. The four points $P[i], P[j], P[k], P[m]$ form a quadrilateral, called the "boundary point" of the pallet. $P[i]$ specifies the first point on the pallet, the line between $P[j]$ and $P[i]$ specifies the row direction of the pallet, the line between $P[k]$ and $P[i]$ specifies the column direction of the pallet, and the last point $P[m]$ specifies the last boundary of the quadrilateral. The four points are connected and divided to get the position of each point on the pallet.
Format	LPallet LPalletNo, $P[i], P[j], P[k], P[m], nRow, nColumn, nLay, LayHeight$;
Parameter	LPalletNo: Serial number of the local pallet variable $P[i], P[j], P[k], P[m]$: Defines the four points of the pallet nRow: Number of rows, value range 1 to 1000 nColumn: Number of columns, value range 1 to 1000 nLay: Number of layers, value range 1 to 1000 LayHeight: Layer height, value range 0 to 2000, in mm

Note:

1. The four points that define the pallet cannot be duplicated.

Example:

LPallet 2, $P[1], P[2], P[3], P[4], 5, 4, 1, 15$;



8.5.19Pallet

(1) Three-point method

Name	Pallet
Function	Pallet definition
Description	Sets global pallet variables. Often used in conjunction with P=Pallet for using simple pallets. The three points $P[i], P[j], P[k]$ form a parallelogram, called the "boundary point" of the pallet. $P[i]$ specifies the first point on the pallet, the line between $P[j]$ and $P[i]$ specifies the

row direction of the pallet, and the line between P[K] and P[i] specifies the column direction of the pallet. The principle of this instruction is to create a pallet boundary based on the three input points, and set the pallet model based on the number of rows, columns, layers, and layer height. In the future, only the information of rows, columns, and layers needs to be used to move to the specified position on the pallet.

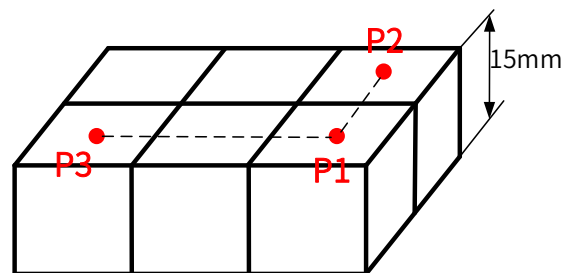
Format	Pallet PalletNo,P[i],P[j],P[k],nRow, nColumn,nLay,LayHeight;
Parameter	PalletNo: The serial number of the global pallet variable, and the value range is 0 to 255 P[i],P[j],P[k]: Defines three points of the pallet nRow: Number of rows, value range 1 to 1000 nColumn: Number of columns, value range 1 to 1000 nLay: Number of layers, value range 1 to 1000 LayHeight: Layer height, value range 0 to 2000, in mm

Note:

1. You can define pallet as a single row or column, but you cannot define a pallet as a point.
2. When the number of pallet layers is greater than 1, any two points are not allowed to be duplicated, otherwise the calculation of pallet points will fail.

Example:

Pallet 1,P[1],P[2],P[3],2,3,1,15;



(2) Four-point method

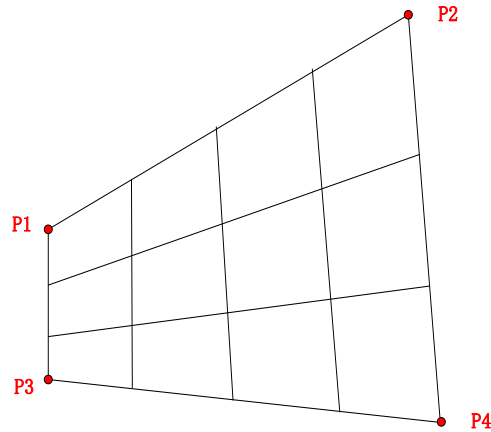
Name	Pallet
Function	4-point definition of pallet
Description	Sets the local pallet variable by 4 points. Often used in conjunction with P=Pallet for using simple pallets. The four points P[i],P[j],P[k],P[m] form a quadrilateral, called the "boundary point" of the pallet. P[i] specifies the first point on the pallet, the line between P[j] and P[i] specifies the row direction of the pallet, the line between P[K] and P[i] specifies the column direction of the pallet, and the last point P[m] specifies the last boundary of the quadrilateral. The four points are connected and divided to get the position of each point on the pallet.
Format	Pallet PalletNo,P[i],P[j],P[k], P[m],nRow, nColumn,nLay,LayHeight;
Parameter	PalletNo: Serial number of the global pallet variable P[i],P[j],P[k],P[m]: Defines the four points of the pallet nRow: Number of rows, value range 1 to 1000 nColumn: Number of columns, value range 1 to 1000 nLay: Number of layers, value range 1 to 1000 LayHeight: Layer height, value range 0 to 2000, in mm

Note:

2. The four points that define the pallet cannot be duplicated.

Example:

Pallet 2,P[1],P[2],P[3], P[4],5,4,1,15;



8.5.20P=Pallet

Function: Gets pallet points

Description: Gets points on the pallet. Used in conjunction with the instruction Pallet, the points in the local pallet variable can be used.

Format 1:

P=Pallet(PalletNo, Row, Column, Lay);

Parameters:

PalletNo: Pallet number, which is the global pallet variable number, and the value range is 0 to 255;

Row: Row index, 0 to 999

Column: Column index, 0 to 999

Lay: Layer index, 0 to 999

Return value:

P: Position variable to save the selected point on the pallet

Format 2:

P=LPallet(PalletNo, Row, Column, Lay);

Parameters:

PalletNo: Pallet number, which is a partial pallet variable number

Row: Row index, 0 to 999

Column: Column index, 0 to 999

Lay: Layer index, 0 to 999

Return value:

P: Position variable to save the selected point on the pallet

Note:

1. The serial number of rows, columns, and layers starts from 0.
2. P = LPallet is the local pallet variable, row, column, layer number must be less than the number

of rows, columns, and layers defined in the instruction LPallet and LPallet4.

3. The coordinate system number, tool number, and user number of the point in the local pallet variable are (3, x, 0), which means that the generated point must be a point in the tool coordinate system, without a user number, with a tool number. The tool number is the same as the first point in the local pallet variable.
4. The coordinate system number, tool number, and user number of the point in the global pallet variable are (3, x, 0), which means the generated point must be a point in the tool coordinate system, without a user number, with a tool number. The tool number is the same as the tool number of pallet calibration point 1.

Example:

```
P[2]=Pallet(1,1,1,0);
```

9 Communication Instructions

9.1 Open Socket

Function: The robot acts as a client and connects to an external server.

Description: Specifies the server IP address and port number for connection to the server.

Format:

Format 1: `Open Socket(IP,SvrPort,ClientPort,BufType);`

Format 2: `Open Socket(IP,SvrPort,ClientPort, BufType,B);`

Parameters:

IP: Server IP address.

SvrPort: Server port number, 1 to 65535

ClientPort: Client port number, range: 2 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number.

BufType: The way to store data in the buffer, supporting circle and single. When set to circle, data is received from the peer after the instruction Open Socket is executed. When the buffer is full, the user will be alerted and no new data will be received.

Circle - When the data is not acquired by Get Port or instruction GetAString, the data in the buffer will be accumulated.

Single - When the data is obtained by Get Port or instruction GetAString, only the data received at that time will be acquired.

B: (Optional) Only B/LB variables are allowed, return value (1 for success, 0 for failure).

Description:

1. Open Socket is used to enable local controller as a client to communicate with external server.
2. Once Open Socket is started, the port remains open. You can close it using the instruction Close Socket or it will be automatically closed when another program is opened manually.
3. During use, loop statements are generally used to perform loop execution, judge the return value parameter and jump out of the loop after successful opening.
4. When the server is not turned on, if format 1 is used, the function will get stuck until the server is turned on; if format 2 is used, the function will not get stuck, and you can tell if the connection is successful by the return value B/LB and the prompt in the notification bar.
5. When connection is made with specified port number (1024-65535), after the communication is disconnected by the instruction Close Socket, or by the communication service management function, or by the peer device, please wait for a period of time (Windows 7 system: 3 to 5s, Windows 10 system: 1 to 2 min) before using the instruction Open Socket to establish a new connection.

Note:

Restrictions on port number: When the robot acts as a client, the port number range is 1024 to 65535. 1111, 2222, 3333, 4444, 5555, 6666, 7777, 8888, 9999, 1217, 8080, 11740, 4003 have special meanings and should be used with caution.

Port number distribution	Port number	Application
0 to 1023 Well known ports	0 to 1023 HTTP 80, Telnet 23, SMTP 25, DNS 53 ModbusTCP 502	Usually the communication over these ports clearly indicates the protocol for a service.
1024 to 65535 Registered port	1111/7777/8888/9999	Reserved for robot system
	1217	codesys application port
	2222	API only
	3333	Teach pendant only
	4444	When a local controller is used as a server and an external device is used as the only client whose port No. is unknown, you can use a port No. 4444.
	5555	Dedicated tooling testing port
	6666	Console debug information output port
	8080	Port 8080 is the same as port 80 and is used for WWW proxy services and can realize web browsing.

Example:

LB[1] = 0;

While LB[1]<>1

Open Socket("192.168.2.5",1025,1026,Single, LB[1]);

EndWhile;

9.2 Open Com

Function: Connects to serial port

Description: Specify the server IP address and port number to connect to the serial port.

Format:

Format 1: Open Com(Port, Baudrate);

Format 2: Open Com(Port, Baudrate,B);

Parameters:

Port: Serial port number (the controller has only one serial port and the serial port number is 1)

Baudrate: Baud rate, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

B: (Optional) Only B/LB variables are allowed, return value (1 for success, 0 for failure).

Description: Once Open Com is started, the port remains open. You can close it using instruction Close Com or it will be automatically closed when another program is opened manually.

Note:

1. Before using this instruction, make sure the user serial port feature is on. (User serial port feature is off by default)

There are two ways to turn on the user serial port feature:

Method 1: (Configure the controller through the teach pendant)

On the connected teach pendant, go to Set > System > Others and open the serial port. When finished, restart the controller.

Method 2: (Configure the controller through SecureCRT software)

a. After connecting to the controller through the SecureCRT software, enter the account: root, password: r

b. Then enter the start command: set_serial 1

c. Restart the controller

2. When the serial port is cyclically opened and closed, please add 4 to 5s interval after the instruction Open Com.

Example:

```
LB[1] = 0;
```

```
While LB[1]<>1
```

```
Open Com(1, 9600, LB[1];
```

```
EndWhile;
```

9.3 Close Socket

Function: Closes the Ethernet port

Description: Closes the Ethernet port and cuts off the Ethernet connection

Format: Close Socket,ClientPort;

Parameters: ClientPort: Client port to be closed

Range: 2 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number.

Description:

Close Socket is used to close an open Ethernet port. It can be used in the following two situations:

(1) Close the connection to the peripheral when the local controller is acting as a client

(2) Close the connection to the peripheral when the local controller is acting as a server

Example:

```
Close Socket,1025;
```

9.4 Close Com

Function: Closes the serial port

Description: Closes the serial port and cuts off the serial port connection

Format: Close Com, Port;

Parameters: Port: Serial port (Controller has only one serial port: 1)

Note: When the serial port is cyclically opened and closed, please add 4 to 5s interval after the instruction Open Com.

Example:

Close Com, 1;

9.5 Send Port

Description: Sends string to far end port of network

Format 1:

Send Port[Port],String;

Parameter:

Port: When using client-server communication, this Port is the client port number.

Range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number.

When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

String: Sent string.

Return value:

StrVar: Saves the string read from the receive buffer

Format 2:

Send Port[Port],String,Type;

Parameters:

Port: When using client-server communication, this Port is the client port number. When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

Type: Type of data to send, which can be String/Binary/Hex. When this parameter is not available, the data is sent as a string by default, equivalent to using String. Where:

String: Data is sent as a string.

Binary: Converts a binary string into a corresponding binary value for sending (temporarily invalid)

Hex: Convert a hexadecimal string to a corresponding hexadecimal value for sending

Return value:

StrVar: Saves the string read from the receive buffer

Description:

1. When using client-server communication, it can be used both when the local controller is the client and when the local controller is the server. However, the port number here is always the client port number.
2. When the local controller acts as a server and an external device acts as the only client whose port number is unknown, you can use port number 4444.
3. When using serial port communication, this Port is the serial port number. Note:
The user serial port function is turned off by default after the system is flashed.
To open the user serial port function, follow the steps below:
 - a. After connecting to the controller through the SecureCRT software, enter the account: root, password: r
 - b. Then enter the start command: set_serial 1
 - c. Restart the controller
4. In TCP/serial communication, you cannot send and receive data to and from the same port/serial port at the same moment in multiple tasks, otherwise communication exceptions will occur.

Example:

```
## Send string "ABC" directly
Send Port[1025 "ABC", String;
## Send global string variable
Str[1] = "ABC";
Send Port[1025]Str[1],String;
## Send local string variable
String ss = "ABC";
Send Port[1025], ss,String;
```

```
##Send hexadecimal data
String Str1;
Str1 = "213A716A3477";
Send Port[1025]Str1, Hex;
```

9.6 Get Port

Function: Receives data from remote port

Description: Within a certain period of time, receive data from the remote port and place it in a buffer. When data is not received after the specified time, jump to the specified label.

Format: Get Port[Port],T[Time],Goto L[Index];

Parameters:

Port: When using client-server communication, this Port is the client port number.

Range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

When using serial communication, this Port is the serial port number (only one serial port, the serial

port number is 1).

Time: Wait time (0 to 65535), unit is s.

Index: The tag number to jump to when the wait times out.

Description:

1. When using client-server communication, it can be used both when the local controller is the client and when the local controller is the server. However, the port number here is always the client port number.
2. When the local controller acts as a server and an external device acts as the only client whose port number is unknown, you can use port number 4444.
3. When using serial port communication, this Port is the serial port number. The user serial port function is turned off by default after the system is flashed.
To open the user serial port function, follow the steps below:
 - a. After connecting to the controller through the SecureCRT software, enter the account: root, password: r
 - b. Then enter the start command: set_serial 1
 - c. Restart the controller
4. Receive port data within time T[Time]. If the reception is successful, the data is stored in the receive buffer and the next line of instruction continues to be executed (without executing Goto L[Index]). If no data is received, execute Goto L[Index], jumping to L[Index].
5. If BufType in the instruction Open Socket is set to circle, the instruction Get Port will get all the data since the last instruction Get Port; if set to single, the instruction Get Port will get the most recently received data. Buffered data will be automatically cleared after acquisition.

Example:

(1) When set to circle:

Scenario 1:

The peer sends "abc" first, then "efg".

In this case, the robot executes the following instructions:

```
Get Port[1025]T[1]Goto L[2];
```

```
Str1 = GetPortbuf(0,10,1025 );
```

Str1 result is "abcefg"

Scenario 2:

The peer sends "abc".

In this case, the robot executes the following instructions:

```
Get Port[1025]T[1]Goto L[2];
```

```
Str1 = GetPortbuf(0,10,1025 );
```

Str1 result is "abc "

The peer sends "efg".

In this case, the robot executes the following instructions:

```
Get Port[1025]T[1]Goto L[2];
```

```
Str1 = GetPortbuf(0,10,1025 );
```

Str1 result is "efg"

(2) When set to single:

Scenario 1:

The peer sends "abc" first, then "efg".

In this case, the robot executes the following instructions:

```
Get Port[1025]T[1]Goto L[2];
```

```
Str1 = GetPortbuf(0,10,1025 );
```

Str1 result is "efg"

9.7 GetPortbuf

Function: Gets string from receive buffer

Description: Reads a specified number of characters from a specified position of a receive buffer and stores them to a specified string variable.

Format: StrVar = GetPortbuf(StartBit , Num, Port);

Parameters:

StartBit: Starting read position of the receive buffer, range: (0 to 1023).

Num: Number of characters read, range (1 to 100); when receive through the serial port, the range is: (1 to 16).

Port: When using client-server communication, this Port is the client port number.

Range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

Return value: StrVar: Saves the string read from the receive buffer

Description:

1. Gets Num consecutive characters from a StartBit in the receive buffer and pass it to the specified string variable. Pass 0 on failure.
2. The instruction GetPortbuf must be used in conjunction with the instruction Get Port, otherwise the acquired string is an empty string, which causes subsequent logic errors.

Example:

```
##Gets three characters from the second bit
```

```
L[2]:
```

```
Get Port[1025]T[1]Goto L[2];
```

```
Str1 = GetPortbuf(2,3,1025);
```

9.8 GetPortState

Function: Gets the status of the specified socket communication connection

Description: Gets the status of the specified socket communication connection

Format: GetPortState(PortNo)

Parameters:

PortNo: Client port number, range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

Return value:

1: Indicates a good connection

0: Indicates a connection exception

9.9 GetSocketNo

Function: Gets the client port number for communication

Description: Gets the client port number for communication

Format: GetSocketNo ("IPAdress",RVar)

Parameters:

IPAdress: The IP address of the external device. In particular, when the IP address is 0.0.0.0, all client port numbers are acquired.

RVar: Only R/LR variables are allowed, used to store the port numbers of all communication clients connected to the specified IP address device. When the controller has multiple connections to external devices, then the port numbers of all connections on the client are stored in an array of variables starting with the R/LR variable.

Return value:

Returns the number of connected clients

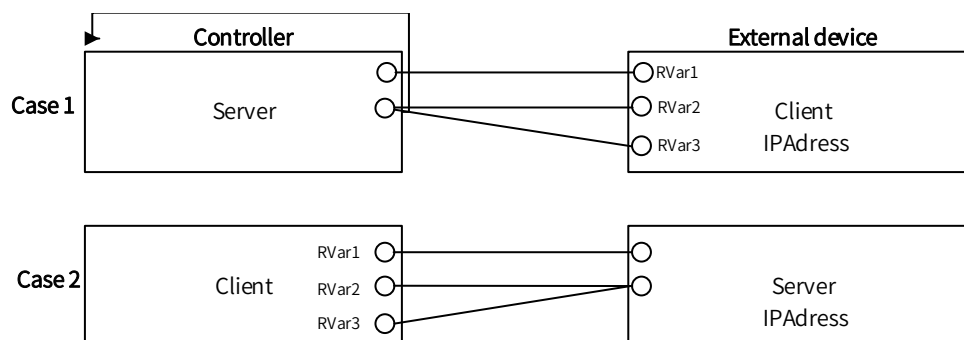
Description:

By specifying the IP address of the external device in the connection between the controller and the external device, you can get the client port number.

Note:

(1) The port number of the client is obtained.

(2) When the controller has multiple connections to external devices, the client port numbers of all these connections are obtained.



(3) The instruction GetSocketNo gets the general communication port number (general communication is established through Set > System > CommSet > ServiceManage, or

communication is established through the instruction Open Socket). It cannot get the special communication port number (such as communication established through Function > VisionCalib).

Example:

```
B[1] = GetSocketNo ("192.168.23.100",R[3] );
```

```
If(B[1]>0)
```

```
    ##Takes the first connection
```

```
    Get Port[R3]T[1]Goto L[1];##Gets data
```

```
EndIf;
```

9.10SetSocketRecvType

Function: Sets the type of data received by the robot

Description: Sets the type of data received by the robot. The received data is saved as a string, binary or hexadecimal string. (Binary is temporarily invalid)

Format: SetSocketRecvType(Port,Type)

Parameters:

Port: Range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

Type: Type of data received, which can be String/Binary/Hex. When this parameter is not available, the data is sent as a string by default, equivalent to using String. Where:

String: Indicates that the data is received as a string.

Binary: Indicates that data is received in binary and converted to a string corresponding to the binary representation (temporarily invalid).

Hex: Indicates that data is received in hexadecimal and converted to a string corresponding to the hexadecimal representation.

(This parameter can also use 0 for String, 1 for Binary, and 2 for Hex)

Description:

1. When using client-server communication, it can be used both when the local controller is the client and when the local controller is the server. However, the port number here is always the client port number.
2. When the local controller acts as a server and an external device acts as the only client whose port number is unknown, you can use port number 4444.
3. Scope of the instruction: The whole system, including the main program, subprogram, and multithread. The instruction is initialized to the String type when the program exits.
4. To switch the receive data type, it is recommended to use the instruction SetSocketRecvType before the instruction Open Socket.

Example:

```
## When an external device sends data in hexadecimal, the robot can receive in two ways:
```

SetSocketRecvType(1025, Hex);	Get Port[1025],T[1],Goto L[2];
Get Port[1025],T[1],Goto L[2];	Str1 = GetPortbuf(2,3,1025);
Str2 = GetPortbuf(2,3,1025);	StrToHex(Str1,Str2);

9.11 GetAString

Function: Receives data from remote port

Description: Receives data from the remote port within a certain period of time, puts it into a buffer, and gets a segment of data truncated by a specified separator. When no data is received within a specified time, jump to a specified tag.

Format: GetAString Port[PortNo],T[Time],Str[Str],Sep[\n],Goto L[Index];

Parameters:

Port: When using client-server communication, this Port is the client port number.

Range: 1 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

When using serial communication, this Port is the serial port number (only one serial port, the serial port number is 1).

Time: Wait time (0 to 65535), unit is s.

Str: Split string

Sep: Separator

Index: The tag number to jump to when the wait times out.

Description:

1. When using client-server communication, it can be used both when the local controller is the client and when the local controller is the server. However, the port number here is always the client port number.
2. When the local controller acts as a server and an external device acts as the only client whose port number is unknown, you can use port number 4444.
3. When using serial port communication, this Port is the serial port number. The user serial port function is turned off by default after the system is flashed.

To open the user serial port function, follow the steps below:

- a. After connecting to the controller through the SecureCRT software, enter the account: root, password: r
 - b. Then enter the start command: set_serial 1
 - c. Restart the controller
4. Receive port data within time T[Time]. If the reception is successful, the data is stored in the receive buffer and the next line of instruction continues to be executed (without executing Goto L [Index]). If no data is received, execute Goto L [Index], jumping to L[Index].
 5. If BufType in the instruction GetAString is set to circle, GetAString gets all the data separated by a delimiter between the last reception and this reception, the data that has been successfully obtained

is deleted from the cache queue and the subsequent newly received data is added to the queue; if to single, GetAString gets the latest received data and clears the previously received data.

Example:

```
GetAString Port[1024],T[1],Str[1], "\n",Goto L[2];
```

Use case:

```
#Open the socket and start receiving data from peer
```

```
Close Socket, 1026;
```

```
LB[1] = 0;
```

```
While LB[1]<>1
```

```
Open Socket("192.168.2.5",1025,1026,Circle,LB[1]);
```

```
EndWhile;
```

```
#Extract data from the cache data queue, start searching for data from the queue header until the separator is found. Store the data from the queue header to the separator in Str [1], delete the data from the cache header to the separator, and move the data after the separator to the queue header.
```

```
L[2]:
```

```
GetAString Port[1024],T[1],Str[1],"\n",Goto L[2];
```

```
Print Str[1];
```

Note:

1. When retrieving data from the cache data queue, the retrieved data includes the separator, for example, in the application case, Str[1] ends with \n.
2. Special treatment is carried out for separator \n, \r, all other characters are parsed according to the actual number of characters.

10 Information Interaction Instructions

10.1 Alarm

Function: Displays custom alarms

Description: Displays user-defined alarm information in the message bar

Format: Alarm [Index];

Parameters: Index: Custom alarm number (0 to 15)

Example:

```
Alarm [2];
```

10.2 Print

Function: Prints information

Description: Prints variables or strings in the message bar

Format: Print [Object];

Parameters: Object: Variable or string that needs to be printed (B\R\D\P\PR\LB\LR\LD\string variable or string).

Due to screen size limitations of the teach pendant, a maximum of 5 expressions can be connected for printing.

Description:

1. The printed content cannot exceed 99 characters.
2. When two instructions PRINT are used continuously, in order to facilitate debugging, delay (typical 0.5 second) can be added when appropriate.

Example:

Print B[1]+P[2]+LR[1];

10.3 GetPlcVarByte

Function: Gets variable values of Byte type

Description: Gets variable values of Byte type from the PLC

Format: GetPlcVarByte(VarIndex);

Parameters: VarIndex: Number of PLC Byte variables (0 to 255)

Return value: The value of the PLC Byte variable read, return value type: Byte

Example:

B[1]= GetPlcVarByte(3) ;

10.4 GetPlcVarInt

Function: Gets variable values of Int type

Description: Gets variable values of Int type from the PLC and stores it

Format: GetPlcVarInt(VarIndex) ;

Parameters: VarIndex: Number of PLC Int variables (0 to 255)

Return value: The value of the PLC Int variable read, return value type: Int

Example:

R[1]= GetPlcVarInt(3) ;

10.5 GetPlcVarDInt

Function: Gets variable values of DInt type

Description: Gets variable values of DInt type from the PLC

Format: GetPlcVarDInt(VarIndex) ;

Parameters: VarIndex: Number of PLC DInt variables (0 to 255)

Return value: The value of the PLC DInt variable read, return value type: Int.

Example:

R[1]= GetPlcVarDInt(3);

10.6 GetPlcVarLReal

Function: Gets variable values of LReal type

Description: Gets variable values of LReal type from the PLC

Format: GetPlcVarLReal(VarIndex)

Parameters: VarIndex: Number of PLC LReal variables (0 to 255)

Return value: The value of the PLC LReal variables read, return value type: double.

Example:

```
D[1]= GetPlcVarLReal(3);
```

10.7 TimeStart

Function: Starts timer

Description: Starts timer for timing

Format: TimeStart(TimerIndex);

Parameters: TimerIndex: Timer number (0 to 4)

Example:

```
TimeStart(0);
```

10.8 TimeOut

Function: Outputs timer duration

Description: Outputs timer duration to specified variable

Format: TimeOut(TimerIndex, Var);

Parameters: TimerIndex: Timer number (0 to 4)

Var: A variable of type double, in which the timer duration (s) is stored

Note:

1. After the timer is turned on, the values continue to accumulate unless the timer is restarted using the TimeStart command;
2. The timer works in the controller and will not be reset due to program switching, unless the timer is restarted using the TimeStart command;
3. A timer that is not started can be compiled, but the value is invalid.

Example:

```
START;
```

Programming example:

```
##Start Timer 1
```

```
TimeStart(1);
```

```
Movj P[1],V[30],Z[0];
```

```
##Read Timer 1
```

```
Timeout(1,D[2]);
```

```
Movj P[2],V[30],Z[0];
```

```
##Read Timer 1
```

```
Timeout(1,D[3]);
```

END;

10.9 GetModBusCoil

Function: Gets coil values from ModBus slave

Description: Gets coil values from ModBus slave and saves them in Byte variables

Format: GetModBusCoil(StartAddr, CoilNum, Var);

Parameters: StartAddr: The starting address (0 to 8191) of the coil in bits.

CoilNum: Number of coils to read (1 to 8).

Var: Byte/ B/LB type variables are used to store the read coil values, and non-Byte type variables are allowed. If it is a non Byte type variable, the obtained Byte type value will be forcibly converted to the corresponding data type for storage.

Note:

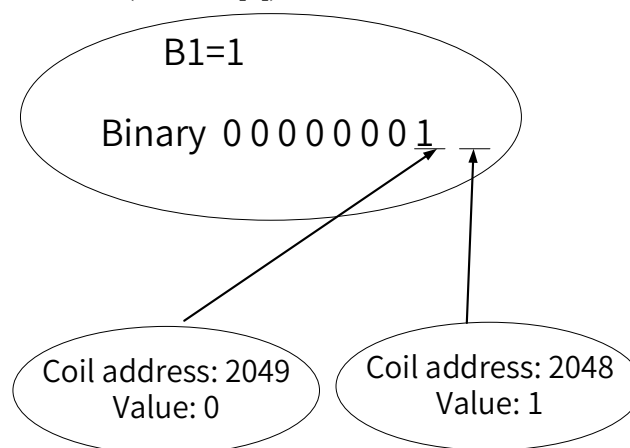
1. The values of one or more ModBus slave coils are read, arranged sequentially from low to high, and stored in a specified variable.
2. The BYTE type variable occupies 8 bits and can accommodate up to 8 coil values.
3. The values are stored from the lower to the higher bits. For example, the coil value of the start address is stored to bit 0 of the variable, the coil value of "start address + 1" is stored to bit 1, and so on.
4. When there are less than 8 coils, the excess coil value is not read and the value on the corresponding bit is taken as 0.

Example:

Task: Read value in coil address 2048, 2049

Read two coils starting from address 2048 and store them in B1

Programming: GetModBusCoil(2048,2,B[1]);



10.10 SetModBusCoil

Function: Sets coil value of the ModBus slave

Description: Sets the Byte type value to the coil value of the ModBus slave

Format: SetModBusCoil(StartAddr, CoilNum, ByteValue);

Parameters: StartAddr: Starting address of the coil (2048 to 4095) || (6144 to 8191), in bits.

CoilNum: Number of coils to write (1 to 8).

ByteValue: Byte type value (0 to 255), according to which the coil value of ModBus slave is set

Description:

1. Byte type value, set from low to high into one or more coils.
2. Byte type value occupies 8 bits and can be set for up to 8 coils.
3. The variable is set from low to high. For example, bit0 is set to the coil at the "start address", bit1 is set to the coil at the "start address+1", and so on.
4. When the number of coils set is less than 8, the value of the extra bit will not sent to the coils.
5. The range of coil address is 2048 to 4095 and 6144 to 8191; coils outside the range will generate errors.

Example:

Task: Set the values in coil addresses 2048, 2049, and 2050 to 1, 0, and 1.

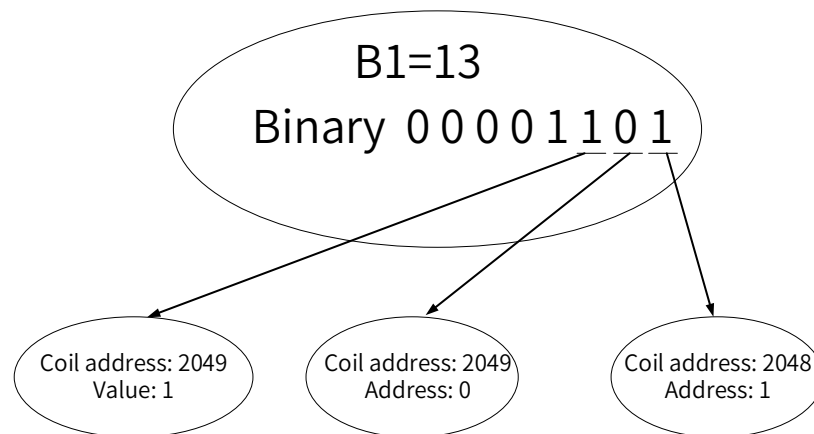
Programming:

```
B[1]=13;
```

```
## Sets B[1] value to three coils with addresses 2048, 2049, 2050
```

```
##In this example, B[1]=5 also works. B[1]=13 is used as an example to show that the extra bits will not be sent.
```

```
SetModBusCoil(2048,3, B[1]);
```



10.11 GetModBusReg

Function: Reads value from register of ModBus slave

Description: Read VarNum data from the ModBus register according to the data type specified by DataType, forcibly convert the data to the data type specified in VarName, and store the data in the variable array specified in VarName

Format: GetModBusReg(StartAddr,VarName,DataType,VarNum);

Parameters: StartAddr: Starting address of the register (0 to 65535)

VarName: Variable name. The read data is stored in an array headed by this variable (B R D L B L R L D arrays only).

DataType: Data type. Read data according to the data type specified below:

1: Short type, occupies 2 bytes, i.e. 1 register

- 2: Int type, occupies 4 bytes, i.e. 2 registers
- 3: Float type, occupies 4 bytes, i.e. 2 registers
- 4: Represents double type, occupies 8 bytes, i.e. 4 registers

VarNum: The number of data read (range: 1 to 256)

Note: Since the data needs to be stored in an array, VarName + the variable number in VarNum <= the length of the array.

Example:

GetModBusReg(16384, R[1],1,2);##Reads two Short-type data (2*1 registers) from the starting address 16384 of the register, and converts them to R variable (int) for storage in R[1] and R[2].

GetModBusReg(16384, R[1]1,2,3,); ##Reads three int-type data (3*2 registers) from the starting address 16384 of the register, and converts them to R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384, R[1]1,3,3,); ##Reads three float-type data (3*2 registers) from the starting address 16384 of the register, and converts them to R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384, R[1]1,4,3,); ##Reads three double-type data (3*4 registers) from the starting address 16384 of the register, and converts them to R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384, D[1],1,2);##Reads two short data (3*1 registers) from the starting address 16384 of the register, and converts them to double variables for storage in R[1], R[2] and R[3].

10.12 SetModBusReg

Function: Sets the value of the ModBus slave register

Format 1: SetModBusReg(StartAddr,Value, DataType,VarNum);

Description: Converts Value to the data type corresponding to DataType, and sets VarNum DataType data to a modbus address with a starting address of StartAddr and a length of VarNum*DataType.

Format 2: SetModBusReg(StartAddr,&VarName, DataType,VarNum);

Description: Converts the data in the variable array specified by VarName to the data type corresponding to DataType, and sets VarNum DataType data to a modbus address with a starting address of StartAddr and a length of VarNum*DataType.

Parameters: StartAddr: Register starting address (16384 to 32767 or 49152 to 65535).

VarName: Starting element of array variable (B R D LB LR LD arrays only)

Value: After the setting is completed, the specified Modbus addresses are set to this value.

DataType: Data type. Convert data according to the data type specified below:

- 1: Short type, occupies 2 bytes, i.e. 1 register
- 2: Int type, occupies 4 bytes, i.e. 2 registers
- 3: Float type, occupies 4 bytes, i.e. 2 registers
- 4: Double type, occupies 8 bytes, i.e. 4 registers

VarNum: The number of data written (range: 1 to 256)

Note: Since the data needs to be stored in an array, VarName + the variable number in VarNum <= the length of the array.

Example:

SetModBusReg(16384, &R[1]1,2);##Converts the values of R [1] and R [2] into short data type and stores them at register addresses from 16384 to 16384+2*1.

SetModBusReg(16384, &R[1]1,2,3);##Converts the values of R [1] and R [2] into int data type and stores them at register addresses from 16384 to 16384+3*2.

SetModBusReg(16384, &R[1]1,3,3);##Converts the values of R [1] and R [2] into float data type and stores them at register addresses from 16384 to 16384+3*2.

SetModBusReg(16384, &R[1]1,4,3);##Converts the values of R [1] and R [2] into double data type and stores them at register addresses from 16384 to 16384+3*4.

SetModBusReg(16384,20,4,3) ##Converts 20 to double data (20.000) and stores it at the register addresses from 16384 to 16384+3*4, where all three data stored are 20.000.

SetModBusReg(16384,R[1],4,3) ##Converts the value of R[1] to double data (20.000) and stores it at the register addresses from 16384 to 16384+3*4, where all three data stored are value of R[1].

11 Gravity Load-Related Instructions

11.1 GripLoad

Function: Sets the ID of work object currently used

Description: Sets the ID of work object currently used. When this instruction is executed, the active object ID will be synchronized to the teach pendant.

Format: GripLoad ObjNo

Parameters: ObjNo: Active object ID, integer value, 0 to 15.

Note:

GripLoad 0 means activating the object 0. Since object 0 is not editable, activating the object 0 is equivalent to unloading the object.

Example:

GripLoad 6;

11.2 SetGripLoadMass

Function: Sets the mass of the object

Description: Sets the mass of the specified object. When this instruction is executed, the set object mass will be synchronized to the teach pendant.

Format: SetGripLoadMass (ObjID, ObjMess)

Parameters: ObjID: Object ID, integer data, 0 to 15

ObjMess: Object mass, double data, in kg

11.3SetGripLoadCog

Function: Sets the centroid of the object

Description: Sets the centroid of the specified object. When this instruction is executed, the set object centroid will be synchronized to the teach pendant.

Format: SetGripLoadCog(ObjID,dx,dy,dz)

Parameters: ObjID: Object ID, integer data, 0 to 15

dx,dy,dz: Centroid offset, double data, in mm

11.4SetGripLoadOrient

Function: Sets the pose of the object

Description: Sets the pose of the specified object. When this instruction is executed, the set object pose will be synchronized to the teach pendant.

Format: SetGripLoadOrient(ObjID, A,B,C)

Parameters: ObjID: Object ID, integer data, 0 to 15

A,B,C: Pose of the object, double data, in degree

11.5SetGripLoadInertia

Function: Sets the inertia of the object around its centroid coordinate system

Description: Sets the inertia of the object around its centroid coordinate system (centroid coordinate system direction is aligned with the end flange coordinate system)

Sets the mass of the specified object. When this instruction is executed, the set object inertia will be synchronized to the teach pendant.

Format: SetGripLoadInertia(ObjID,Ix,Iy,Iz);

Parameters: ObjID: Object ID, integer data, 0 to 15

Ix,Iy,Iz: Rotating inertia around the x, y, z axis, double data.

11.6SetToolMass

Function: Sets the mass of the tool

Format: SetObjMass(ToolID,ToolMass)

Parameters: ToolID: Tool ID, integer data, range (0 to 15)

ToolMass: Tool mass, double data, unit kg

11.7SetToolCog

Function: Sets the centroid of the tool

Description: Sets the centroid of the tool

Format: SetToolCog(ToolID,dx,dy,dz)

Parameters: ToolID: Tool ID, integer data, range (0 to 15)

dx,dy,dz: Tool centroid offset, double numeric value or variable, in mm.

11.8SetToolOrient

Function: Set the pose of the tool

Format: SetToolOrient(ToolID, A,B,C);

Parameters: ToolID: Tool ID, integer data, range (0 to 15)

A,B,C: Pose of the object, double data, unit °

11.9SetToolInertia

Function: Sets the inertia of the tool

Format: SetObjInertia(ToolID, A,B,C)

Parameters: ToolID: Tool ID, integer data, range (0 to 15)

A,B,C: Rotating inertia, double data, units (***)

12 Current Protection-Related Instructions

12.1AvgCurLmt

Function: Configures average load rate limit

Description: Sets whether the average load rate detection is valid and the average load rate. Can be applied to single and all axes.

Format: AvgCurLmt (Enable,AxisNo,ratio);

Parameters: Enable: Specifies valid/invalid by integer data. 1 for valid, 0 for invalid;

AxisNo: Specifies the axis number by integer data. Exception: All axes can be used as objects by '0'.

ratio: Uses integer data to specify the local limiting factor for the average load rate of a single axis, in units of 1%, value range (1, 150). This parameter is invalid if Enable is 0.

Properties of detection switch:

1. The main switch on the setting interface of the teach pendant defaults to "True", and the default will be restored when the robot restarts;
2. The detection switch for each single axis in the instruction defaults to "True", and the default local limit coefficient for a single axis is 100.
3. To restore the default parameters, see Section 1.4.

12.2MaxTrqLmt

Function: Configures the maximum torque limit (current limit)

Description: Sets whether current detection is active and current limit parameters in the program, which can be applied to single and all axes.

Format: MaxTrqLmt (Enable,AxisNo,ratio);

Parameters: Enable: Specifies valid/invalid by integer data. 1 for valid, 0 for invalid;

AxisNo: Specifies the axis number by integer data. Exception: All axes can be used as objects by '0'.

Ratio: Use integer data to specify the local limiting factor for the current of a single axis, in units of 1%, value range (1, 150). This parameter is invalid if Enable is 0.

Properties of detection switch:

1. The main switch on the setting interface of the teach pendant defaults to "True", and the default will be restored when the robot restarts;
2. The detection switch for each single axis in the instruction defaults to "True", and the default local limit coefficient for a single axis is 100.
3. To restore the default parameters, see Section 1.4.

13 CollisionDetection-Related Instructions

13.1 SetCollMode

Function: Sets the status of collision detection switch in play mode and the actions triggered after collision alarm

Description: Sets the status of collision detection switch in play mode and the actions triggered after collision alarm. The parameter set by this instruction takes effect together with the parameter set in the user interface of the teach pendant (The alarm triggered actions are solely subject to the parameter set by the instruction.)

Format: SetCollMode(value,iMode)

Parameters: value: You can select ON to turn on collision detection, or OFF to turn off collision detection.

iMode: Selects the action triggered after the collision alarm, integer data, 0 to 7, currently only 3 is supported. If other integer values are entered, they will be processed as 3.

Note:

1. This instruction only works in Play mode.
2. For the initialization conditions, see [1.4 Scope of Variables and Instructions](#)

Example:

```
SetCollMode (ON,3);
```

13.2 SetAxisCollMode

Function: Sets the single-axis collision detection switch in play mode

Description: Sets the collision detection switch for the single axis in the play mode. The parameter set by this instruction takes effect together with the parameter set in the user interface of the teach pendant.

Format: SetAxisCollMode(iAxisNo, value):

Parameters: iAxisNo: Axis number, integer data, range 1 to 6.

value: The collision detection switch flag, ON or OFF.

Note:

1. This instruction only works in Play mode.
2. For the initialization conditions, see [1.4 Scope of Variables and Instructions](#)

Example:

```
SetAxisCollMode(2, OFF)
```

13.3 SetAxisCollLevel

Function: Sets the single-axis collision detection switch in play mode

Description: Sets the collision detection sensitivity of the single axis in the play mode. The parameter set by this instruction takes effect together with the parameter set in the user interface of the teach pendant, that is, the actual sensitivity is obtained by multiplying the two parameters. The greater the sensitivity, the less likely it is to detect a collision. The sensitivity here is the same as that set in the teach pendant, ranging from 25 to 300. Assuming the sensitivity set in the teaching pendant is a , and the sensitivity set in the instruction is $iPara_Level$, then the final sensitivity $c = a * iPara_Level / 100$, and c must also meet $c \in [25, 300]$. If the calculated value of c exceeds the range, the maximum value of 300 or the minimum value of 25 should be taken accordingly.

Format: `SetAxisCollLevel(iAxisNo,iPara_Level)`

Parameters: $iAxisNo$: Axis number, integer data, range: 1 to 6

$iPara_Level$: Axis collision detection sensitivity in play mode, real number, 25 to 300.

Note:

1. This instruction only works in Play mode.
2. For the initialization conditions, see [1.4 Scope of Variables and Instructions](#)

Example:

`SetAxisCollLevel (1, 157.258)`

14 Interference Area-Related Instructions

Eight additional interference zones can be set by instruction. They are independent relative to the interference zones configured in the user interface, and can take effect simultaneously and independently of the interference zones configured in the teach pendant.

Interference zones can be defined as rectangular, spherical, cylindrical, and articular, respectively, by the instructions WZBoxDef, WZSphDef, WZCylDef, WZLimJDef. The instruction WZDOSet defines the action after interference, activates the interference area, and enables interference detection. The instructions WZEnable and WZDisable activate and deactivate the interference area, respectively.

Note:

The interference area defined by the instruction takes effect only in play mode.

When the project is recompiled, the interference areas defined by the instruction will be emptied.

You can use instructions to repeatedly define the same interference area, and subsequent definitions will replace the previous ones.

The interference area detection is only applicable to detect the end of the robot body end, not the tools for the time being.

After the robot enters the interference zone, due to the inability to stop immediately and the need to decelerate, it is recommended to set the interference area larger than the actual one.

When switching from play mode to teach mode, to ensure that the robot can move out of the interference area normally, the interference area defined by the instruction will be deactivated and no interference area detection will be performed. To ensure that the interference area defined by the instruction takes effect again, it is recommended to return to the starting line to execute the program when switching back from teach mode to play mode.

14.1 WZBoxDef

Function: Defines a rectangular interference area

Description: Defines the rectangular interference area in the base coordinate system using the two diagonal points of the rectangular parallelepiped

Format: WZBoxDef Box[AreaID], nType,P1,P2;

Parameters: AreaID: Interference area ID, integer data (0 to 7)

nType: Type of interference area, 0 means internal interference area and 1 means external interference area

P1 and P2: Diagonal points of the rectangular parallelepiped

14.2 WZSphDef

Function: Defines a spherical interference area

Description: Defines the spherical interference area in the base coordinate system using the spherical center and radius of the sphere

Format: WZSphDef Box[AreaID], nType,P,Radius;

Parameters: AreaID: Interference area ID, integer data (0 to 7)
nType: Type of interference area, 0 means internal interference area and 1 means external interference area
P: Spherical position variable for spherical interference area
Radius: double type data, defining the radius of the spherical interference area

14.3 WZCylDef

Function: Defines a cylindrical interference area
Description: Define the cylindrical interference area in the base coordinate system using the center and radius of the bottom surface of cylinder, and the cylinder height
Format: WZCylDef Box[AreaID], nType,P,Radius,Height;
Parameters: AreaID: Interference area ID, integer data (0 to 7).
nType: Type of interference area, 0 means internal interference area and 1 means external interference area
P: Position variable of the center of the bottom surface of cylinder
Radius: double type data, defining the radius of the bottom surface of the cylindrical interference area
Height: double type data, defining the height of the cylindrical interference area

14.4 WZLimJDef

Function: Defines a joint interference area
Description: Define the interference area in the joint coordinate system
Format: WZLimJDef Box[AreaID], nType,PR[1],PR[2];
Parameters: AreaID: Interference area ID, integer data (0 to 7)
nType: Type of interference area, 0 means internal interference area and 1 means external interference area
PR[1], PR[2]: Two joint limits, one small and one large. The translation variable contains the angles of each joint, and if the joint angles of the two translation variables are the same, the joint limits are not limited, for example, they both can be 0.
Note: For a joint interference area, an interference action is triggered whenever one axis enters the interference area.

14.5 WZDOSet

Function: Defines actions after interference and activates interference area
Description: Defines actions after interference, such as setting variables, setting signals, or generating alarms. It also activates an interference area.
Format: WZDOSet Box[AreaID], output signal;
Parameters: AreaID: integer data, constant, range 0 to 7.
Output signal: Includes B variable |OUT[***]|Alarm[***];

Note:

1. When the instruction WZDOSet defines an interference action, it also activates this interference area by default, which is equivalent to enabling the instruction WZEnable.
2. When the interference area is activated but no interference action is defined using WZDOSet, the user defined alarm 0 is used by default when interference occurs.
3. For a joint interference area, an interference action is triggered whenever one axis enters the interference area.

Example:

```
WZDOSet Box[1], B[2];  
WZDOSet Box[1], Out[3];  
WZDOSet Box[1]Alarm[4];
```

14.6 WZDisable

Function: Disables interference area

Description: Deactivates the specified interference area and turns off interference detection for this interference area

Format: WZDisableBox[AreaID];

Parameters: AreaID: Interference area ID, integer data (0 to 7)

14.7 WZEnable

Function: Enables interference area

Description: Activates the specified interference area to turns on interference detection for this interference area

Format: WZEnable Box[AreaID];

Parameters: AreaID: Interference area ID, integer data (0 to 7)

Note: When the interference area is activated but no interference action is defined using WZDOSet, the user defined alarm 0 is used by default when interference occurs.

15 Conveyor-Related Instructions

15.1 CnvVision

Function: Turns on/off the vision port of the conveyor

Description: Opens/closes the vision port of the specified conveyor. This instruction is to be used after the instruction Open and before the instruction GetCnvObject.

Format: CnvVision(Conveyor[Index],ON|OFF,ClientPort);

Parameters: Index: Conveyor number (0 to 3)

ON|OFF: Turns on or off the vision port

ClientPort: Client port number

Range: 2 to 5 or 1024 to 65535

2 to 5 indicate the first to fourth Socket connection, and the client port number does not need to be specified;

1024 to 65535 indicate a Socket connection with a specified client port number;

Example:

```
CnvVision (Conveyor[0],ON,1024);
```

```
GetCnvObject(0,0), Goto L[0];
```

```
.....
```

```
CnvVision (Conveyor[0],OFF,1024);
```

Note:

1. When a vision port is opened using the instruction CnvVision, Get Port, Send Port and SetSocketRecvType cannot be used for that port.
2. This instruction is used to open the port to trigger visual photography. Therefore, if you switch modes when running the dynamic follower, the port will be closed. When switching back to the play mode, you need to open the port again to trigger visual photography simply by going back to the start line and running the instruction again.

15.2 RefSys

Function: Switches the coordinate system

Description: When external motion mechanisms are used, such as conveyors or tables, it is necessary to identify which coordinate system the position variables are in. In this case, use RefSys to switch to the external coordinate system; when it is necessary to use the position variables in the robot coordinate system, use RefSys again to switch back to the robot coordinate system.

Format 1: RefSys Base;

Description:

1. Switch to the base coordinate system. When you switch back from the conveyor or table coordinate system to the robot coordinate system, motion stops.
2. After the instruction RefSys Base is executed, the current tool number of the system is updated to 0.

Format 2: RefSysConveyor(ConveyIndex, Tool);

Description:

1. Switch to the conveyor coordinate system. Since the conveyor is a dynamic coordinate system, the TCP synchronizes with the conveyor after this instruction is executed.
2. After the instruction RefSys Base is executed, the current tool number of the system is updated to the tool number written by the Tool parameter.

Parameters:

ConveyIndex: Conveyor index, 0 to 3

Tool: Tool number, indicating synchronized conveyor movement at the specified TCP.

Format 3: RefSysWorkBench(ConveyIndex, Tool);

Description: Switches to the rotary table coordinate system. Since the rotary table is a dynamic coordinate system, the TCP moves synchronously with the table after this instruction is executed.

Parameters:

ConveyIndex: Conveyor index, 0 to 3

Tool: Tool number, indicating synchronized conveyor movement at the specified TCP.

Description:

Note: The instruction PE is not valid when conveyor following process is used. That is, PE parameter must not be included in the motion instructions during the conveyor following process.

Example:

START;

CnvVision (Conveyor[1],ON,1025); ##Opens vision port for conveyor 1, client port number 1025

P[30]=(0,0,10,0,0,0),(0,0,0,0),(7,0,0); ##Defines P[30] as 10 mm directly above the object on conveyor 1

L[0]:

Movj P[0],V[30],Z[0],Tool[2];

GetCnvObject(1,0) Goto L[0]; ##Receives data of object type 0 on conveyor 1

RefSys Conveyor(1,Tool[2]); ##Uses the end of tool 2 to complete the synchronized movement with conveyor 1

Movl P[30],V[100],Z[1],Tool[2]; ##P[30] is a point in the coordinate system of conveyor 1

Set Out[1],ON,T[0]; ##Turns on the switch to suck objects

Delay T[1];

RefSys Base; ##Switches to robot coordinate system

Jump P[1],V[100],Z[0],Tool[2],LH[10],MH[-750],RH[10]; ##P[1] is a point in the robot coordinate system

Set Out[1]OFF,T[0]; ##Places the object

Delay T[1];

Goto L[0];

CnvVision (Conveyor[1],OFF,1025);

END;

15.3 GetCnvObject

Function: Receives position data of objects entering the grip space on the conveyor

Description: Gets data of a specified type of object from a specified conveyor, the object is removed from the queue

Format: GetCnvObject(CnvID, ObjID),Goto L[Index];

Parameters: CnvID: Conveyor number (0 to 3)

ObjID: Object type number (0 to 15)

Index: Label number to jump to if the reception is not successful within the specified time. If the reception is successful, jump to the next line.

Description:

1. The instruction GetCnvObject is used with the instruction CnvVision when the detection method is vision. When the vision port is opened, the received vision data is transmitted directly to the DSP for processing until the vision port is closed.
The instruction GetCnvObject is used separately when the detection method is sensor.
2. If the position data of the object is received, the next line of instruction continues to be executed (without executing Goto L [Index]). If no data is received, execute Goto L [Index], jumping to L[Index].
3. Calling GetCnvObject in the loop can continuously get the position data of the objects on the conveyor that enter the grip space. Each time GetCnvObject is used, it will get a piece of position data to get the position of the object on the conveyor belt that matches the object type. The next time it is used, it will automatically get the next piece of position data to get the position of the next object on the conveyor belt that matches the object type.
4. The instruction GetCnvObject/CopyCnvObject/ClearCnvObject needs to match the conveyor number in the instruction CnvVision when the detection method is vision.

Note:

When the camera or sensor acquires the positions of the objects on the conveyor, the positions will all be saved in an area from where the instruction GetCnvObject fetches them.

Example:

START;

L[0]:

##Open the port. External device as server, address 10.44.53.13, port number 1025;

##Local controller as client, port number 1026.

Open Socket("10.44.53.13",1025,1026,B[0]);

If B0 == 0

Goto L[0];

CnvVision (Conveyor[1],ON,1026);

P[30]=(0,0,10,0,0,0),(0,0,0,0),(7,0,0); Defines P[30] as 10 mm directly above the object on conveyor 1

L[1]:

Movj P[0],V[30],Z[0],Tool[2];

GetCnvObject(1,0), Goto L[1]; ##Receives data for object type 0 on conveyor 1

RefSys Conveyor(1,Tool[2]); ##Uses the end of tool 2 to complete the synchronized

```

movement with conveyor 1
Movl P[30],V[100],Z[1],Tool[2];      ##P[30] is a point in the coordinate system of conveyor 1
Set Out[1],ON;                        ##Turns on the switch to suck objects
Delay T[1];
RefSys Base;                          ##Switches to robot coordinate system
Jump P[1],V[100],Z[0],Tool[2],LH[10],MH[-750],RH[10];    ##Moves object to P[1]
Set Out[1],OFF,T[0];                 ##Places objects
Delay T[1];
Goto L[1];
CnvVision (Conveyor[1],OFF,1026);
Close Socket,1026;
END;

```

15.4 CopyCnvObject

Function: Copies visual data of objects on conveyor

Description: Copies the object of the specified type on the specified conveyor, with the object remaining in the queue

Format: CopyCnvObject(CnvID, ObjID),Goto L[Index];

Parameters: CnvID: Conveyor number (0 to 3)

ObjID: Object type number (0 to 15)

Index: Label number to jump to if the reception is not successful within the specified time. If the reception is successful, jump to the next line.

Description:

1. The instruction GetCnvObject/CopyCnvObject/ClearCnvObject needs to match the conveyor number in the instruction CnvVision.

Example:

```
CopyCnvObject(1, 0),Goto L[1];
```

15.5 ClearCnvObject

Function: Clears object from conveyor

Description: Clears one or all objects on the specified conveyor. Detected conveyor objects are not automatically cleared when the program is paused or stopped, and need to be programmed with this instruction if clearance is required

Format: ClearCnvObject(CnvID,Num|ALL);

Parameters: CnvID: Conveyor number (0 to 3)

Num|ALL: Number of objects to clear; if ALL, all objects will be cleared

Description:

1. The instruction GetCnvObject/CopyCnvObject/ClearCnvObject needs to match the conveyor number in the instruction CnvVision.

Example:

```
##Remove two object data from conveyer 1  
ClearCnvObject(1,2);  
##Remove all object data from conveyer 2  
ClearCnvObject(1,ALL);
```

16 Position Latch Instructions

The position latch function is described below.

Use the user I/O Out[14] or Out[15] signal to trigger the system to save the position of the robot at the trigger moment, and call the corresponding instruction to obtain the position when needed.

This instruction is used in the function of capturing images of object in high speed motion. After position latch is turned on, the user I/O Out[14] or Out[15] is used to trigger image capture and record the current position during the motion. During the subsequent motion, the final position that the robot should reach is calculated simultaneously based on the latched information, and the robot ultimately moves to this point.

Note:

- 1) IRCB500 and IRCB300 series controllers support the position latch function. The user only needs to connect the corresponding I/O to the camera as a trigger to take photos.
- 2) This feature is not supported for IRCB10 series controllers.
- 3) The IRCB300 series controllers support triggering position latch through one output signal (Out15). In order to avoid the problem that the position latch function is not available due to damage, the IRCB500 series controllers support triggering position latch through two signals. The user can choose to trigger position latch by user I/O Out14 or Out15 according to needs, and the default is the user I/O Out[14].

16.1 LatchEnable

Function: Turns on/off position latch function

Description: When position latch is enabled, the system monitors the specified signal (Out[14] or Out[15]) for changes. When it detects a change from OFF to ON, the robot position is latched. The coordinates of the latched position can be read through GetLatchPos.

Format: LatchEnable ON|OFF;

Parameters: ON|OFF: ON, OFF

Example:

START;

LatchEnable ON;

ClearLatchPos;

Movl P[0],V[30],Z[0],Tool[0];

Movl P[1],V[30],Z[0],Tool[0],NWait,Out(15,OFF,D[0]),Out(15,ON,D[50]);

B[1]=0;

While B[1]==0

 B[1]=GetLatchPos(P[10],2,0,0);

EndWhile;

Print P[10];

LatchEnable OFF;

End;

Note:

To turn off the position latch function, execute the instruction LatchEnable OFF.

16.2 ClearLatchPos

Function: Clears latched positions

Description: Clears previously latched robot positions

Format: ClearLatchPos;

Parameters: ON|OFF: ON, OFF

Example: See LatchEnable

16.3 GetLatchPos

Function: Reads latched positions

Description: Reads last latched robot position

Format: GetLatchPos(P, CoordID, ToolID, UserID);

Parameters: P: Location variable, in which obtained results are stored

CoordID: Coordinate system number of the acquired position variable (1 to 4)

ToolID: Tool number of the acquired position variable (0 to 15)

UserID: User number of the acquired position variable (0 to 15)

Return value: Returns 0 if no latched data and 1 if latched data, the value of P is valid when 1 is returned

Description:

1. The robot controller supports caching up to 10 latched data. When the latch signal is triggered, the data in the cache is increased by 1. Use the instruction GetLatchPos to decrease the cache data by 1. The cache data will be cleared when the robot is reset.
2. When multiple latched data is triggered, the instruction GetLatchPos reads the latched position sequentially.

In the following example, the position stored in P10 is the latch position triggered when the motion reaches P[0], and the position stored in P11 is the latch position triggered when the motion reaches P[1].

```
START;
```

```
LatchEnable ON;
```

```
ClearLatchPos;
```

```
Movl P[0],V[30],Z[0],Tool[0],NWait,Out(15,OFF,D[0]),Out(15,ON,D[50]);
```

```
Movl P[1],V[30],Z[0],Tool[0],NWait,Out(15,OFF,D[0]),Out(15,ON,D[50]);
```

```
B[1]=0;
```

```
While B[1]==0
```

```
    B[1] =GetLatchPos(P[10],2,0,0);
```

```
EndWhile;
```

```
Print P[10];
```

```
B[2]=0;
```

```
While B[2]==0
```

```
    B[2] =GetLatchPos(P[11],2,0,0);
```

```
EndWhile;
```

```
Print P[11];  
LatchEnable OFF;  
End;
```

3. For initialization conditions, see: [1.4 Scope of Variables and Instructions](#)

17 Application Cases

17.1 Communication Settings

Vision instructions are commands used by the controller to communicate with external vision devices, either through a network or serial port.

There are two forms of network port communication, depending on the usage.

(1) A local controller is used as a server and an external device is used as a client:

The robot controller as server is opened by default when the system starts up, and there is no need to open it with the instruction Open Socket in the program. However, you need to set a port number and configure the connection with this port on the vision device. Go to Set > System > CommSet.

(2) A local controller is used as a client and an external device is used as a server:

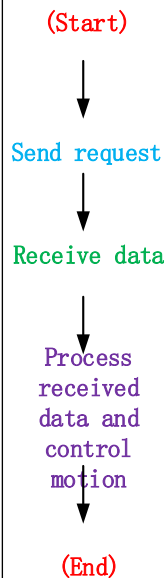
The robot controller is used as a client, and the IP of the connection needs to be selected in the program using the instruction Open Socket.

Robot as client: Set port to 1026 in instruction
External vision as server: Set server port to 1025 in CommSet

```
START;
PR0=(0,0,10,0,0,0);
TxBuf = "TA";
RxBuf = "";
L[2]:
Movj P[0],V[30],Z[0];
L[0];
Open Socket("10.44.53.13",1025,1026,B0);
If B0 == 0
    Goto L[0];
EndIf;
Send Port[1026],TxBuf;
L[1]:
Get Port[1026], T[10], Goto L[1];
RxBuf = GetPorbuf(0,100,1026);
B1 = StrGetData(RxBuf,"#",P10);
Cnvt(P[10],P[20],World);
Movl Offset(P[20],PR0),V[30],Z[0];
Set Out[1],ON,T[0];
Delay T[1];
Jump P[1],V[100],Z[0],LH[10],MH[-750],RH[10];
Set Out[1],OFF,T[0];
Delay T[1];
Goto L[2];
Close Socket,1026;
END;
```

Robot as server: Set server port to 1025 in CommSet
External vision as client: Set port to 1026

```
START;
TxBuf = "TA";
RxBuf = "";
L[2]:
Movj P[0],V[30],Z[0];
Send Port[1026],TxBuf;
L[1]:
Get Port[1026], T[10], Goto L[1];
RxBuf = GetPorbuf(0,100,1026);
B1 = StrGetData(RxBuf,"#",P10);
Cnvt(P[10],P[20],World);
Movl Offset(P[20],PR0),V[30],Z[0];
Set Out[1],ON,T[0];
Delay T[1];
Jump P[1],V[100],Z[0],LH[10],MH[-750],RH[10];
Set Out[1],OFF,T[0];
Delay T[1];
Goto L[2];
END;
```



For serial communication, you can directly program after connecting the serial port, as shown in the following example:

```
START;
B[0] = 0;
While B[0] <> 1
```

```

Open Com(1,19200,B[0]) ;
EndWhile;
Send Port[1],"TA";
L[1]:
Get Port[1],T[0],Goto L[1];
Str[1] =GetPortbuf(0,100,1);
Print Str[1];
Close Com, 1;
END;

```

For the follow-up process, you can open the vision port of the conveyor using the instruction CnvVision. The vision data is automatically transferred to the controller and processed after the vision port is opened. The dynamic follow-up and grip action can be achieved after acquiring the data of the object on the conveyor, and the vision port of the conveyor is closed after the grip is successful. The instruction flow is Open ->CnvVision(ON) ->GetCnvObject -> Movl P ->CnvVision(OFF) ->Close. A simple programming example is as follows:

```

START;
L[0]:
##Open the port. External device as server, address 10.44.53.13, port number 1025;
##Local controller as client, port number 1026.
Open Socket("10.44.53.13",1025,1026,B[0]);
If B[0] == 0
Goto L[0];
EndIf;
CnvVision (Conveyor[1],ON);
##Defines P[30] as 10 mm directly above the object on conveyor 1. Note that the coordinate system
number is 7, which is a point that follows the conveyor movement.
P[30]=(0,0,10,0,0,0),(0,0,0,0),(7,0,1);
L[1]:
Movj P[0],V[30],Z[0],Tool[0];
GetCnvObject(1,0), Goto L[1];      ##Receives data for object type 0 on conveyor 1
RefSys Conveyor[1];               ##Uses coordinate system of conveyor 1
Movl P[30],V[100],Z[1],Tool[0];   ##Moves to P[30]
Set Out[1],ON;                    ##Turns on the switch to suck objects
Delay T[1];
RefSys Base;                      ##Switches to robot coordinate system
Jump P[1],V[100],Z[0],Tool[0],LH[10],MH[-750],RH[10];    ##Moves object to P[1]
Set Out[1]OFF,T[0];               ##Places the object
Delay T[1];
Goto L[1];
CnvVision (Conveyor[1],OFF);
Close Socket,1026;
END;

```