INOVANCE

Advancing industrial technology,
for a better world

Industrial Robot

Instruction Guide

Industrial Automation | Intelligent Elevator | New Energy Vehicle | Industrial Robot | Rail Transit

Data code 19012825A00

Legal and Disclaimer

Copyright

Trademarks

INOVANCE is a registered trademark of Shenzhen Inovance Technology Co., Ltd. and its affiliates. All other trademarks or registered trademarks mentioned in this documentation are the property of their respective owners. Unauthorized use of these trademarks by third parties for any purpose without written authorization could violate the rights of their owners.

Disclaimer of Liability

Due to continuous updates and improvements of products and technologies, the content of this documentation may not fully match the actual products. In the event of any discrepancies, the actual products shall prevail.

The contents are subject to change without notice due to product upgrade.

Waste Disposal

The storage, use, and disposal of this product (including optional accessories) must comply with local laws and regulations.

Qualified Personnel

The product/system described in this documentation may be operated only by personnel qualified for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions.

Qualified personnel can identify the risks of the product/system and prevent possible dangers.

Proper Use of the Product

Proper transportation, storage, assembly, installation, commissioning, operation, and maintenance are required to ensure the safe operation of the product without any problems. The required ambient conditions must be met. All operations must follow the guidelines provided in this documentation.

Preface

Introduction

The "Industrial Robot Instructions Guide" (This guide) serves as a reference for programming Inovance Technology industrial robots. It is primarily intended for engineers, operators, and system integrators, aiming to provide comprehensive instruction descriptions, instruction application specifications, and other content.

More Data

Data Name	Data Code	Description
Industrial Robot Instructions Guide (This guide)	19012825	This guide introduces the function, format, parameter specifications, usage examples, and application cases of program instructions for Inovance Technology industrial robot.

Revision History

Revision Date	Document Version	Description
2026-03	A00	First issue

Access to the Guide

This guide is not delivered with the product. You can obtain the PDF version by the following methods:

- Do keyword search under Service and Support at https://www.inovance.com/global.
- Scan the QR code on the product with your smart phone.
- Scan the QR code below to install My Inovance app, where you can search for and download user guides.

Basic Safety Instructions

Safety Precautions

Safety Disclaimer

- This chapter presents essential safety instructions for a proper use of the equipment. Before operating the equipment, read through the guide and comprehend all the safety instructions. Failure to observe the safety instructions may result in death, personal injuries, or equipment damage.
- "DANGER", "WARNING", and "CAUTION" items in this guide do not indicate all safety precautions that need to be followed; instead, they just supplement the safety precautions.
- Use this product according to designated environment requirements. Damage caused by improper use is not covered by warranty.
- Inovance shall take no responsibility for any personal injuries or property damage caused by improper usage.

Safety Levels and Definitions

Indicates that failure to comply with the notice will result in severe personal injuries or even death.

Indicates that failure to comply with the notice may result in severe personal injuries or even death.

Indicates that failure to comply with the notice may result in minor or moderate personal injuries or equipment damage.

Safety Instructions

- Drawings in the guide are sometimes shown without covers or protective guards. Remember to install the covers or protective guards as specified first, and then perform operations in accordance with the instructions.
- The drawings in the guide are shown for illustration only and may be different from the product you purchased.
- Operators must take mechanical precautions to protect personal safety and wear protective equipment, such as safety shoes, safety clothing, safety glasses, protective gloves, and protective sleeves.

Unpacking
 • Do not install the equipment if you find damage, rust, or signs of use on the equipment or accessories upon unpacking.

- Do not install the equipment if you find water seepage or missing or damaged components upon unpacking.
- Do not install the equipment if you find the packing list does not conform to the equipment you received.

- Before unpacking, check whether the outer package is intact, damaged, wet, damped, or deformed.
- Unpack the package in sequence. Do not hit the package violently.
- Check whether there is damage, rust, or injuries on the surface of the equipment and equipment accessories before unpacking.
- Check whether the package contents are consistent with the packing list before unpacking.

Storage and Transportation

- Large-scale or heavy equipment must be transported by qualified professionals using specialized hoisting equipment. Failure to comply may result in personal injuries or equipment damage.
- Before hoisting the equipment, ensure the components such as the front cover and terminal blocks are secured firmly with screws. Loosely-connected components may fall off and result in personal injury or equipment damage.
- Never stand or stay below the equipment when the equipment is being hoisted by the hoisting equipment.
- When hoisting the equipment with a steel rope, ensure the equipment is hoisted at a constant speed without suffering from vibration or shock. Do not turn the equipment over or let the equipment stay hanging in the air. Failure to comply may result in personal injuries or equipment damage.

CAUTION

- Handle the equipment with care during transportation and mind your steps to prevent personal injuries or equipment damage.
- When carrying the equipment with bare hands, hold the equipment casing firmly with care to prevent parts from falling. Failure to comply may result in personal injuries.
- Store and transport the equipment based on the storage and transportation requirements. Failure to comply will result in equipment damage.
- Avoid storing or transporting the equipment in environments with water splash, rain, direct sunlight, strong electric field, strong magnetic field, and strong vibration.
- Avoid storing the product for more than 3 months. When the product needs to be stored for an extended period, take more strict protection and necessary inspection.
- Pack the equipment strictly before transportation. Use a sealed box for long-distance transportation.
- Never transport the equipment with other equipment or materials that may harm or have negative impacts on this equipment.

Installation

DANGER

- The equipment can only be operated by professionals with electrical knowledge. Non-professionals are not allowed to operate on the equipment.

WARNING

- Read through the guide and safety precautions before installation.
- Do not install the equipment in places with strong electric or magnetic fields.
- Before installation, check that the mechanical strength of the installation site can bear the weight of the equipment. Failure to comply will result in mechanical hazards.
- Do not wear loose clothes or accessories during installation. Failure to comply may result in an electric shock.
- When installing the equipment (such as controller) in a closed environment (such as a cabinet or casing), use a cooling device (such as a fan or air conditioner) to cool the environment down to the required temperature. Failure to comply may result in equipment over-temperature or a fire.
- Do not retrofit the equipment.
- Do not loosen the fixing bolts of the product parts and components.
- When the equipment (such as controller) is installed in a cabinet or final assembly, a fireproof enclosure providing both electrical and mechanical protections must be provided. The IP rating must meet IEC standards and local laws and regulations.
- Before installing devices with strong electromagnetic interference, such as a transformer, install a shielding device for the equipment to prevent malfunction.

- Install the product on an incombustible object such as metal and do not touch or attach the product to combustibles. Failure to comply can result in fire accident.

- Cover the top of the equipment (such as controller) with a piece of cloth or paper during installation. This is to prevent unwanted objects such as metal chippings, oil, and water from falling into the equipment and causing faults. After installation, remove the cloth or paper on top of the equipment to prevent over-temperature caused by poor ventilation due to blocked ventilation holes.

Wiring

- Equipment installation, wiring, maintenance, inspection, or parts replacement must be performed by professionals.
- Before wiring, disconnect all the power supplies of the equipment. Wait for at least the time designated on the equipment warning label before further operations because residual voltage still exists after power-off. Measure the direct voltage of the main circuit and ensure that the voltage is within the safety range. Failure to comply can result in an electric shock.
- Do not perform wiring, remove the equipment cover, or touch the circuit board with power ON. Failure to comply can result in an electric shock.
- Check that the equipment is grounded properly. Failure to comply can result in an electric shock.

- Do not connect the input power supply to the output side of the equipment. Failure to comply will result in equipment damage or even a fire.
- When connecting a drive to the motor, check that the phase sequences of the drive and motor terminals are consistent to prevent reverse motor rotation.
- Cables used for wiring must meet cross sectional area and shielding requirements. The shield of the cable must be reliably grounded at one end.
- Fix the terminal screws with the tightening torque specified in the user guide. Improper tightening torque may overheat or damage the connecting part, resulting in a fire.
- After wiring is done, check that all cables are connected properly and no screws, washers or exposed cables are left inside the equipment. Failure to comply may result in electric shock or equipment damage.

- Follow the proper electrostatic discharge (ESD) procedure and wear an anti-static wrist strap to perform wiring. Failure to comply may result in damage to the equipment or to the internal circuit of the product.
- Use shielded twisted pairs for the control circuit. Connect the shield to the grounding terminal of the equipment for grounding purpose. Failure to comply can result in equipment malfunction.

Power-On

- Before power-on, check that the equipment is installed properly with reliable wiring and the motor can be restarted.
- Check that the power supply meets equipment requirements before power-on to prevent equipment damage or a fire.
- Do not open the cabinet door or protective cover of the product, touch any wiring terminal of the product, or remove any part of the product with power on. Failure to comply can result in an electric shock.

- After the completion of wiring and parameter setting, perform a robot test run to confirm that the robot can operate safely. Failure to comply may result in personal injuries or equipment damage.
- Before power-on, check that the rated voltage of the equipment is consistent with that of the power supply. Failure to comply can result in a fire.
- Before power-on, check that no one is near the equipment, motor, or machine. Failure to comply may result in death or personal injury.

Operation

- The equipment must be operated only by professionals. Failure to comply can result in death or personal injury.
- Do not touch any connecting terminals or disassemble any unit or component of the equipment during operation. Failure to comply can result in an electric shock.

- Do not touch the equipment enclosure, fan, or resistor with bare hands to sense the temperature. Failure to comply may result in personal injury.
- Prevent metal or other objects from falling into the equipment during operation. Failure to comply may result in a fire or equipment damage.

Maintenance

- Equipment installation, wiring, maintenance, inspection, or parts replacement must be performed by professionals.
- Do not maintain the equipment with power ON. Failure to comply can result in an electric shock.
- Before maintenance, cut off all the power supplies of the equipment and wait for at least the time designated on the equipment warning label.
- In case of a permanent magnet motor, do not touch the motor terminals immediately after power-off because the motor terminals will generate induced voltage during rotation even after the equipment power supply is off. Failure to comply can result in an electric shock.

- Perform routine and regular inspection and maintenance on the equipment in accordance with maintenance requirements and keep a record.

Repair

- Equipment installation, wiring, maintenance, inspection, or parts replacement must be performed by professionals.
- Do not maintain the equipment with power ON. Failure to comply can result in an electric shock.
- Before inspection and repair, cut off all the power supplies of the equipment and wait for at least the time designated on the equipment warning label.

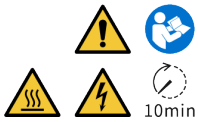
- Submit a request for repair services in accordance with the product warranty agreement.
- When the fuse is blown or the circuit breaker or earth leakage current breaker (ELCB) trips, wait for at least the time designated on the equipment warning label before power-on or further operations. Failure to comply may result in death, personal injuries or equipment damage.
- Troubleshooting and repair must be performed by professionals in accordance with the repair instructions, with repair records kept properly.
- Replace quick-wear parts of the equipment in accordance with the replacement instructions.
- Do not operate on damaged equipment. Failure to comply may result in death, personal injuries, or severe equipment damage.
- After the equipment is replaced, check the wiring and set parameters again.

Disposal

- Dispose of retired equipment in accordance with local regulations and standards. Failure to comply may result in property damage, personal injury, or even death.
- Recycle retired equipment in accordance with industry waste disposal standards to avoid environmental pollution.

Safety labels

For safe operation and maintenance, comply with the safety labels on the equipment. Do not damage or remove the safety labels. See the following table for descriptions of the safety labels.

Safety labels	Description
	• Read through the safety instructions before operating the equipment. Failure to comply may result in death, personal injuries, or equipment damage. • Do not touch the terminals or remove the cover with power ON or within 10 min after power-off. Failure to comply can result in an electric shock.

Industrial Information Security

The product provides an interface to connect to the network and transmits data through the network interface. To protect factories, systems, machines, and networks from cyber attacks and ensure safe operation, a proper industrial information security protection mechanism must be implemented.
The customer is responsible for providing and continuously ensuring a secure connection between the product and the customer network or any other network to prevent unauthorized access to its factories, systems, machines, and networks. The system or machine can be connected to the corporate network or the Internet only when secure connections are ensured and appropriate security measures are taken (for example, anti-virus software or firewall installed).
Inovance continuously develops and improves products and solutions to improve safety. It is strongly recommended that you update the product promptly and always use the latest version.

Malware (such as viruses, Trojans, and worms) can put the product in an unsafe running state, which may result in death, serious injuries, and property damage. Observe the following precautions:

- Always use the latest software version. If the product version is no longer supported or the latest version is not used, customers will be exposed to increased risk of cyber attacks.
- Take appropriate protection measures (including but not limited to installing anti-virus software, firewall, WAF, IPS/IDS, and situation awareness system, and applying authentication, data encryption) to prevent files in mobile storage devices from being damaged by malware, and to prevent products, networks, systems, and interfaces from unauthorized access, interference, intrusion, data leakage, or information breach.
- Check all safety-related interfaces and settings after commissioning.

Contents

1 Revision History

Version	Change Description
V4R24C4	1. Added system diagnosis instructions: SetDiagnosisItem, GetDiagnosisItem, SetDiagnosisSave, GetDiagnosisSave. 2. Added bus instructions: SetIOBusState, GetIOBusState. 3. Added interrupt instructions: ISigAD, ISigDA, ISigTimer.
V4R24C3	1. Added interrupt instructions: IConnect, IActive, IDeactive, IEnable, IDisable, IDelete, ISigIn, ISigOut, StopMoving, Trap, EndTrap. 2. Added digitalization instructions: UserLogOut, SetUserLogPath. 3. Added black box instruction: SetVarRecordMode. 4. Added system parameter instruction: GetRobotType. 5. Added collision detection instruction: SetCollDist.
V4R24C2	1. Function return value instruction: Return 2. Func instruction: Added available function return type following Func. 3. Alarm instruction: Alarm can be followed by EasyGo parameter 4. Signal processing instructions: BindTcpSpeed, CloseBindTcpSpeed 5. Dynamic IP acquisition instruction: GetDynamicIP 6. Interference zone-related instructions: SetInterferZone, SetInterferTool 7. System parameter-related instructions: SetJumpSLMode
V4R24C1	1. Global points saving instruction: SavePoints 2. Multi-point recipe instructions: LoadPoints, GetCurPointsFileName 3. The number of timer instructions (TimeStart, TimeOut) was increased from "0 to 4" to "0 to 99". 4. Pallet instruction: Pallet. Added format: Pallet (pallet number, pallet position number) 5. Current position acquisition instruction: GetCurPos. Added format: GetCurPos (Tool[vaule],Wobj[value]). 6. MessageBox instruction: UIDialog 7. Soft float instructions: SoftModeAct, SoftModeDeact, and SoftFollow 8. Safety I/O drop-protection instructions: GrabClose, GrabOpen 9. Free curve instructions: Movs, Curve, CurveParm, EndCurve 10. Constant instruction: Const
S03.23R	1. Added independent axis instructions: IndCMove, IndReset, and IndSpeed 2. Added dynamic tracking instructions: CnvTrigVis, GetCnvObjOutNum, GetCnvObjData, GetCnvObjType, GetCnvCntPos, GetCnvSpeed 3. Added local controller time acquisition instruction: GetLocalSysTime

Version	Change Description
	4. Added singularity crossing instruction: SingAreaHandle 5. Added the temperature rise protection instruction: GetEncoderTemper
S03.22R	1. Point definition change: P/LP variable is changed to position point. Tool number, user number, and frame number is no longer included in the point. Added the external axis data. The P/LP variable represents the actual Cartesian coordinate position in space. The value of the tool end in the specified workobject frame is always recorded when getting the point. When the robot runs a point, it is needed to specify which frame the coordinates are in and which tool to use to approach the point. In addition, the struct members are changed. For details, see the 2.1 Variables section. 2. Changed the keyword of the user frame from "User" to the keyword "Wobj" of the workobject frame. 3. Changed the struct of Tool, Wobj and Load, as described in the 2.1 Variables section. 4. Changed the offset variable PR to the offset variable of the Cartesian frame, and deleted the joint offset. 5. Added the joint variable P/LP, see the 2.1 Variables section. 6. Delete the following instructions: SetToolParm, SetUserParm, SetGripLoadMass, SetGripLoadCog, SetGripLoadOrient, SetGripLoadInteria, SetToolMass, SetToolCog, SetToolOrient, SetToolIntertia, OffSetUserParm, Cnvrt OffSetJ. 7. For the Tool, Wobj and Load variables, their values are changed not using function settings, but instead direct assignment. For eaxample, SetToolMass is changed to Tool[*].TLoad.Mass = 30. 8. Changed SetUserNo to SetSysWobjNo. 9. Changed the WZLimJDef parameter PR to JP. 10. Added the joint motion instruction MovAbsJ, fast pickup instruction CnvFastCatch, coordinate conversion instructions CalFixWobjPos, CalRobholdWobjPos, JToP, PToJ, CVToRobPos, and current point acquisition instruction GetCurPos, GetCurJPos, etc. 11. Changed the meaning of Offset and OffSetT instructions. 12. Added the Speed parameter to the motion instruction, and added linear speed control. For details, see Speed instruction. 13. Changed the specifications of the Msft instruction specification. It can only calculate the offset variable of two position points. 14. PE indicates only the position point. OffSet and OffSetT can only use P (including PE). 15. Changed the specifications of the Clear Tool/Wobj/Load instructions. For details, see the corresponding instructions.

2 Overview

2.1 Variables

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

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/JP/SPEED/PALLET/custom variables	Prefix custom variables with the modifier Global
Global constants	ONCE = 1, ON = 1, OFF = 0, TRUE = 1, FALSE = 0	Use keywords to represent constants
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 value type:

Variable type	Variable data type
Numerical variables	B/LB/R/LR/D/LD/BOOL/BYTE/INT/FLOAT/DOUBLE
Constants	ONCE = 1, ON = 1, OFF = 0, TRUE = 1, FALSE = 0
I/O variables	IN/OUT/INB/OUTB/INW/OUTW
String variables	STR/String
Offset variables	PR/LPR
Position variables	P/LP
Joint position variables	JP
Pallet variables	Pallet/LPallet
Speed variables	V/SPEED
Tool variables	Tool
Workobject variables	Wobj

Variable type	Variable data type
Load variables	Load

- Numerical variables

Variable type	Width	Value range	Subscript range
BOOL	1 bytes	TRUE, FALSE	Custom variables
BYTE	1 bytes	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

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.

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 occurs.

For the assignment of the system variables B, RD, LB, LR, LD, the value is checked against the value range. When the range is exceeded, an error occurs. When B, R, D, LB, LR, or LD is passed into the function as a parameter, forced conversion is carried out.

- I/O variables

Variable type	Character unit	Length	Subscript range	Value range	Remarks
IN	Bit	1Bit	0 to 13823	ON(1)/OFF(0)	

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

   Bytewise reading:

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

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

   Wordwise 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] = InW[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

> CAUTION
>
> Only INB/INW/OutB/OutW 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)

> CAUTION
>
> 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;

InW[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:

Operator	Name	Example
&	Bitwise AND	OUT[0] & OUT[1]
|	Bitwise OR	OUT[0] | OUT[1]
^	Bitwise XOR	OUT[0] ^ OUT[1]
~	Bitwise NOT	~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	Data Type	Length	Subscript range
STR	Char	256 bytes	0 to 255
String	Char	256 bytes	Custom variables

- Offset variables

Variable type	Data Type	Length	Subscript range
PR	Byte	96 bytes	0 to 255
LPR	Byte	96 bytes	0 to 255

- Position variables

Variable type	Data Type	Length	Subscript range
P	RobPos struct	112 bytes	0 to 9999
LP	RobPos struct	112 bytes	0 to 9999

- Joint position variables

Variable type	Data Type	Length	Subscript range
JP	JPos struct	112 bytes	0 to 9999

- Pallet variables

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

- File handle variables

Variable type	Character unit	Length	Subscript range
FileHandle	-	-	Custom variables
FileDirHandle	-	-	Custom variables

- Speed variables

Variable type	Data Type	Length	Subscript range
V	Double	8 bytes	0.1 to 100
Speed	Speed struct	40 bytes	1 to 15000

- Tool variables

Variable type	Data Type	Length	Subscript range
Tool	ToolData	-	0 to 15

- Wobj variables

Variable type	Data Type	Length	Subscript range
Wobj	WobjData	-	0 to 15

- Load variables

Variable type	Data Type	Length	Subscript range
Load	LoadData	-	0 to 15

Example:

Int asd;#Variable definition

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

2.2 Arrays

Supports defining 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 10,000. The total number of elements in an array cannot exceed 10,000.

- **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

2.3 Structs

- **Format**:

Struct StructVariableName

Definition of struct elements

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 variables P/LP, JP, PR/LPR, TOOL, and WOBJ.

The member variables of the P/LP struct are as follows:

```
Struct RobPos
{
    double X;
    double Y;
    double Z;
    double A;
    double B;
    double C;
    int ArmType[4];
    double E1;
    double E2;
    double E3;
    double E4;
    double E5;
    double E6;
}
```

The struct members are accessed as follows: P[1].X P[1].Y P[1].ArmType[0] P[1].E1

The member variables of the JP struct are as follows:

```
Struct RobPos
{
```

```
    double J1;
    double J2;
    double J3;
    double J4;
    double J5;
    double J6;
    double E1;
    double E2;
    double E3;
    double E4;
    double E5;
    double E6;
}
```

The struct members are accessed as follows: JP[1].J1 JP[1].E1

The member variables of the PR/LPR struct are as follows:

```
Struct Pose
{
    double X;
    double Y;
    double Z;
    double A;
    double B;
    double C;
}
```

The struct members are accessed as follows: PR[1].X LPR[1].Y

The member variables of the Tool struct are as follows:

```
Struct ToolData
{
    Bool RobHold;
    Pose TFrame;
    LoadData TLoad;
}
```

The data type of member TFrame is Pose. See Pose structure for details.

The data type of TLoad is LoadData. See LoadData structure for details, as shown below:

```
Struct LoadData
{
    double Mass;
    double Cog[3];
    double Ori[3];
    double Inertial[3];
}
```

The struct members are accessed as follows: Tool[1].RobHold Tool[1].TFrame.X

Tool[1].TLoad.Mass, etc.

The member variables of the Wobj struct are as follows:

```
Struct WobjData
{
    Bool RobHold;
    Bool UFFix;
    Pose UFrame;
    Pose OFrame;
}
```

The data type of the member UFrame is Pose. See Pose struct for details.

The struct members are accessed as follows: Wobj[1].RobHold Wobj[1].UFrame.X

The member variables of the Speed struct are as follows:

```
Struct Speed
{
    double TCP;##TCP speed
    double Ori;##TCP orientation speed
    double Leax;##External axis linear speed
    double Reax;##External axis rotation speed
    int bStatic;##Whether speed is affected by global percentage 1 = Affected 0 = Not Affected
}
```

The struct members are accessed as follows: Speed[100].TCP

Example: Use of custom structs

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

Int var1;#Definition of struct member variables. 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 structure 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;

2.4 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 initialized	Whole system
System variables Tool Wobj Load	Retentive at power off, no initialization	Not initialized	Whole system
Global position variables P/ JP	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 on	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/Close, or when closed or opened through the user interface.	Null data in buffer when disconnected	Whole project
Socket and 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 value		Project compilation	Velset OFF	Between Velset value and Velset OFF or next Velset value in the current program file.	No	No
Velset Rate[value]		Not initialized	-	Takes effect upon execution of the instruction. Velset OFF is not valid for the current instruction.	Yes	No
Group IG OG		Power on initialization	IG OG in undefined state	Whole system	Yes	No
SlewMode		Project compilation	0	Current task	No	No
Motion parameter	SetAccRamp	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	Acceleration jerk: 50.0 Deceleration jerk: 50.0	Whole project	No	No
	SetStopCat1 AccRamp	Re-power on	Level: 100	Whole project	No	Yes
	SetStopCat2 AccRamp	Re-power on	Level: 100	Whole project	No	Yes
	StopCat1Cat 2AccMinLmt Enable	Re-power on	OFF	Whole project	No	Yes
Transition parameter	SetFlyWait	Project compilation, or return to the	0-FLY_FREE	Whole project	No	No

Instruction		Initialization conditions	Initialized value	Scope	Is effective in multitasking?	Is subject to forced blocking?
		start line of the main task, or return to the start line of all tasks	Position transition stress: 100 Orientation transition stress: 100			
Current limit	AvgCurLmt	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	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 tasks	OFF by default, latched positions cleared	Whole project	No	Yes
	ClearLatchPos					
	GetLatchPos					No
Collision detection	SetCollMode	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	Value: ON mode: 0	Whole project	No	No
	SetAxiscollMode		axisno: all value: on			Yes
	SetAxisCollLevel		axisno: all level: 100			
	SetCollDist		Dist: Matches the set value in Auto mode interface			No
Gravity Load-Related Instructions	GripLoad	Retentive at power off, no initialization	-	Whole project	Yes	Yes
Coordinate system	SetSysToolNo		-	Whole project	No	Yes

Instruction		Initialization conditions	Initialized value	Scope	Is effective in multitasking?	Is subject to forced blocking?
number setting	SetSysWobjNo	Retentive at power off, no initialization				
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
	SetJumpSLMode	Project compilation, or return to the start line of the main task, or return to the start line of all tasks	mode: SL_Auto	Whole project	No	Yes
System Diagnostics	SetDiagnosisItem	Retentive at power off, no initialization	Globally effective after setup	Whole project	Yes	Yes
	GetDiagnosisItem		-			Yes
	SetDiagnosisSave		-			Yes
	GetDiagnosisSave		-			Yes

Instruction		Initialization conditions	Initialized value	Scope	Is effective in multitasking?	Is subject to forced blocking?
Bus	SetIOBusState	Retentive at power off, no initialization	Globally effective after setup	Whole project	Use in main task	Yes
	GetIOBusState		-	Whole project	Use in main task	Yes
Robot accuracy mode	SetAccuracyMode	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
Wait		-	-	Whole project	Yes	Yes
WaitInPos		-	-	Current program	No	Yes
Motion instruction with SLOn/SLOff/SLReset parameters		-	-	Whole project	No	Yes

Instruction	Initialization conditions	Initialized value	Scope	Is effective in multitasking?	Is subject to forced blocking?
BindTcpSpeed CloseBindTcpSpeed	Sync project, return to start line, return all to start line, set startup of start line	-	Whole system	Yes	No
GrabClose GrabOpen	-	-	Whole project	No	No

Note:

1. After modifying the .prj/program file/label/global point file, do one of the following:
   1) Switch the control permission.
   2) Start debugging.
   3) Perform single-step debugging.
   4) Click start in auto mode.
   5) Set the start line.
   6) Return to the start line.
   7) Switch to the auto mode.

   Modifying the project file includes:
   1) Modifying any project file in InoRobotLab.
   2) Modifying any project file on the teach pendant and immediately switching connection to InoRobotLab before synchronization of the modification to the robot controller is triggered.
2. In the Sync project to controller dialog, check any of the options including Labels, Global points, and Program file.
3. Go to "Project configuration" > "Save".
4. After modifying the .prj/program file/label/global point file, do one of the following:
   1) Switch the control permission.
   2) Switch from Edit mode to Auto mode.
   3) Switch from Edit mode to Debug mode.
   4) Click the "Save all" button.
   Or click the "Save" button on the corresponding project file page.
   1) Go to "Programming" > "Label" or open the project and go to "Labels", modify the parameters and click "Save".
   2) Open the project and go to "Project" > "Program" > "Instruction", modify the program file and click "Save".
   3) Open the project and go to "Project" > "Program" > "LP[***]", modify the local points and click "Save".
   4) Go to "Programming" > "P[***]", modify the global points and click "Save".
5. Import a global point file.

6. Create, delete, rename, paste, import a program file successfully (or modify the program file and .prj file).
7. Double click another project in the project list to switch the project.
8. Import a project and the name of the imported project is the same as the name of the currently active project.
9. Rename the currently active project.

The following situations will trigger forced blocking:

For an 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 Movj instruction includes "Nwait", the system tool number is switched to "5" only after point P[0] is reached.

2.5 Operators

2.5.1 Arithmetic Operators

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

2.5.2 Logical Operators

Operators	Name
&&	AND
‖	OR
!	NOT

2.5.3 Relational Operators

Operators	Name
==	Equality
>	Greater than

Operators	Name
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Inequality

2.5.4 Bitwise Operators

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

2.5.5 Special Symbols

Operators	Name
##	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

3 Instructions

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-F	G-L	M-R	S-Z
A: ABS, ACC, ACOS, AD, ALARM, ALL, AND, ANGLETORAD, ARMCHANGE, ARMTYPE, ASIN, ATAN, AVGCURLMT	**G:** GET, GETACTIVEGRIPLOADNO, GETACTIVETOOLNO, GETALARMNO, GETAT, GETCNVOBJECT, GETCURPOS, GETLATCHPOS, GETCURPOINTSFILENAME, GETCNVSPEED, GETCNVCNTPOS, GETCNVOBJOUTNUM, GETCNVOBJDATA, GETDYNAMICIP, GETMODBUSREG, GETPLCVARBYTE, GETPLCVARDINT, GETPLCVARINT, GETPORTBUF, GETPORTSTATE, GETRUNSTATE, GETSOCKETNO, GETTORQUE, GLOBAL, GOTO, GRIPLOAD, GETDIAGNOSISITEM, GETDIAGNOSISSAVE, GETIOBUSSTATE, GETAPOINTFROMFILE, GETCURJPOS, GETCNVOBJTYPE, GETMODBUSCOIL, GETPLCVARLREAL, GETSTRASCII, GROUP, GRABCLOSE, GRABOPEN	**M:** MAX, MAXTRQLMT, MH, MID, MIDACCURACY, MIDLEVEL, MIN, MOVABSJ, MOVC, MOVFROMGET, MOVFROMPUT, MOVJ, MOVL, MOVW, MOVTOGET, MOVTOPUT, MSFT, MOVS	**S:** SAVEPOINTFILE, SAVEPOINTS, SEND, SET, SETACC, SETACCRAMP, SETACCURACYMODE, SET AXISCOLLLEVEL, SETAXISCOLLMODE, SETCOLLMODE, SETFLYWAIT, SETINTERFERZONE, SETI NTERFERTOOL, SETMODBUSCOIL, SETMODBUSREG, SETPORTBUF, SETSOCKETRECVTYPE, SE TSYSTOOLNO, SETSYSWOBJNO, SETDIAGNOSISITEM, SETDIAGNOSISSAVE, SETIOBUSSTATE, SINGAREAHANDLE, SIGNAL, SIN, SLDATACLEAR, SLEWMO DE, SLOFF, SLON, SLRESET, SLVSMODE, SOCKET, SOCKETLISTEN, SPRINTF, SQRT, START, STEP, STR, STRCMP, STRFIND, STRFINDEND, STRGETDAT A, STRING, STRLEN, STRREPLACE, STRREVERSE, STRTOASCII, STRTOD, STRTOHEX, STRTOR, STRUCT, SWITCH, SYSTEM, SOFTMODEACT, SOFTMODEDEACT, SOFTFOLLOW, SETUSERLOGPATH, STOPMOVING, SETCOLLDIST, SETVARRECODEMODE

A-F	G-L	M-R	S-Z
B: BASE, BINARY, BINDTCPSPEED, BIT, BOOL, BOX, BREAK, BTOASCII, BYTE, BINDTCPSPEED	**H:** HALT, HEX, HEXTOSTR, HIGHACCURACY, HIGHLEVEL, HOME	**N:** NAME, NORMAL, NOT, NOTES, NWAIT	**T:** TAN, TCP, THEN, TIME, TIMEOUT, TIMESTART, TO, TOOL, TOOLNO, TRIMLEFT, TRIMRIGHT, TRUE, TRAJECTORY, TOOLNO, TRAP, TCP_3P, TCP_3P_ZX
C: CAPS, CASE, CALFIXWOBJPOS, CALROBHOLDWOBJPOS, CLEAR, CLEARCNVOBJECT, CLEARLATCHPOS, CLOSEBINDTCPSPEED, CNVVISION, CNVFASTCATCH, COM, CNVTRIGVIS, COLL_DEFAULT, COLL_REBOUND, COLL_STOP, COPYCNVOBJECT, CORNER, COS, CP, CURRENT, CUSTOMFUNC1, CUSTOMFUNC2, CUSTOMMOVE, CLOSE, CONTINUE, CONVEYOR, CRDNO, CVDATATOPOSE, CVTOROBPOS, CRDNO, CONST, CURVE, CURVEPARM, CLOSEBINDTCPSPEED, CALCULATETOOL, CALCULATEWOBJ, CAL_UF, CAL_OF	**I:** IF, IG, IN, INB, INCLUDE, INCR, INDCMOVE, INDRESET, INDSPEED, INT, INVERT, INW, ICONNECT, IACTIVE, IDEACTIVE, IENABLE, IDISABLE, IDELETE, ISIGIN, ISIGOUT, ISIGAD, ISIGDA, ISIGTIMER	**O:** OBJ, OFF, OFFSET, OFFSETT, OG, ON, OPEN, OR, OUT, OUTB, OUTW, ONCEANDERRNOSTOP	**U:** UNLOAD, UNTIL, UIDIALOG, USERLOGOUT
D: DA, DATA, DEC, DECR, DEFAULT, DELAY, DIST, DO, DOUBLE, DS, DTOSTR, DATA	**J:** J1, J2, J3, J4, J5, J6, J7, J8, JOINT, JUMP, JUMPL, JUMPSLMODE	**P:** PALLET, PAUSE, PE, PICKV, PORT, POW, PR, PRINT, PROGRAMINFO, PTOSTR, PTP, PULSE	**V:** VELSET, VERSION, VOID, VXCALIB
E: EASYGO, ELSE, ELSEIF, END, ENDFOR, ENDFUNC, ENDIF, ENDPROGRAMINFO,	**K:** KIND	**Q:** QUIT	**W:** WAIT, WOBJ, WAITINPOS, WHILE, WORKBENCH, WORLD,

A-F	G-L	M-R	S-Z
ENDSTRUCT, ENDSWITCH, ENDWHILE, EOFFSOFF, EOFFSON, ECAT, EIP, ENDCURVE, ENDTRAP, ECP_KT_1P, ECP_KT_1P_ZX, ECP_UKT_3P, ECP_UKT_3P_ZX			WRITEAPOINTTOFILE, WOBJ_3P, WOBJ_ROT
F: FALSE, FIN, FINE, FLOAT, FLY_FIX, FLY_FREE, FOR, FUNC	**L:** LATCHENABLE, LB, LD, LEFT, LH, LOAD, LOADPOINTFROMFILE, LOADPOINTS, LOWCASE, LOWLEVEL, LOWSPEEDHIGHACCURACY, LP, LPALLET, LPR, LR, LOGIC	**R:** RADTOANGLE, RAPIDMOVE, RATE, REFSYS, REPEAT, RESUME, RET, RETURN, RH, RIGHT, ROBOTNAME, RTOSTR	**X:** XQT
-	-	-	**Y:** -
-	-	-	**Z:** ZR
-	-	-	-

4 Motion Instructions

4.1 Movj

- **Description**: Quickly moves the robot from point to point. The trajectory is usually not a straight line, and all axes reach the target positions simultaneously.
- **Format**: Movj P, V, Z, Tool, [Wobj], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)…],[SLOn/ SLOff/SLReset].
- **Parameter:**

P: Target point in Cartesian space, required.

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

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

PE can be used in Offset and OffsetT. For example, OffSet(PE,PR) indicates that the robot offsets from the current position. See the PE Description for details.

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

Pallet point getting form: Pallet(PalletNo, Row, Colunm, Lay), LPallet(PalletNo, Row, Colunm, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number. For details, see 9.5.20 P=Pallet.

V: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. The Speed instruction is not supported.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the transition length is equal to the full motion length of the instruction.

(2) If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.

(3) If the current instruction is Jump/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].

CAUTION

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length of the instruction to ensure that the two transition areas no longer overlap. If both transition areas at the beginning and end of the instruction are larger than half the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see Transition Characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended Use of Tool Parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended Use of Wobj Parameters.

When not specified, the default Wobj[0] is used.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag.

Indicates to execute the subsequent non-motion instructions immediately before the target point is reached.

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

NOTE

- If the subsequent instruction is a motion instruction, regardless of whether this motion instruction includes NWait, the motion instruction will be issued in advance.
- In the case where multiple motion instructions are used, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance when the first motion instruction is executed, regardless of whether the first motion instruction includes NWait.

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:

Program 1:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [6], ON

End;

Program 2:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Set Out [6], ON

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];

Movl P [5], V[30], Z[0], Tool [0], Wobj[0];

Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [7], ON

End;

Program 3:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Set Out [6], ON

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];

Movl P [5], V[30], Z[0], Tool [0], Wobj[0];

Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [7], ON

End;

In Program 1, 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.

In Program 2, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

In Program 3, Set Out[7] will be executed in advance, and in fact, both Set Out[6] and Set Out[7] are executed at the fifth 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 to the end.

- **Subparameter:**

In: Input signal, only supports In[0]-In[127].

Value: Input signal value, ON or OFF

Example of use:

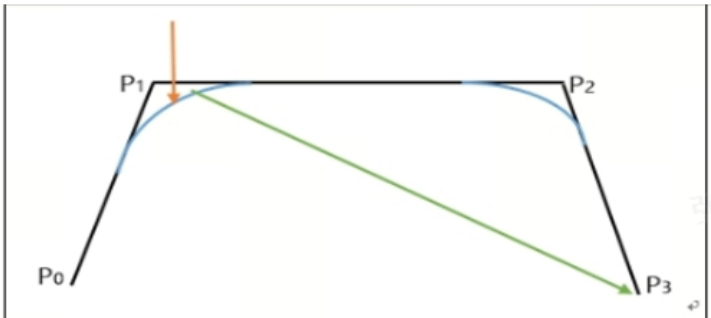
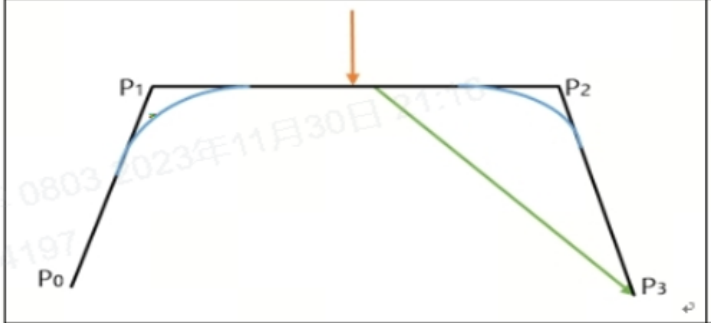
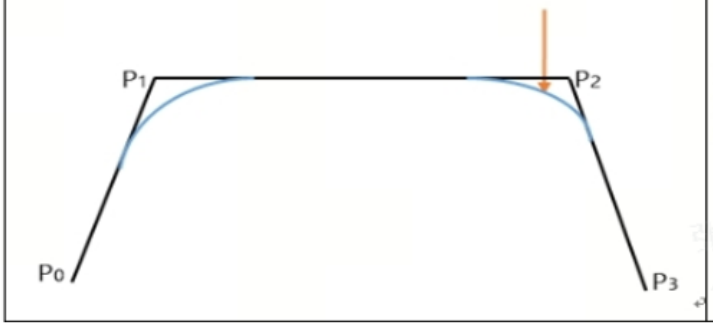
##Detect In[5] during motion, and immediately stop the motion when it is ON; Otherwise, keep moving to the end.

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

CAUTION

- If the Until parameter is used, Z is considered Z[0].
- When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will be executed in advance.
- The stop triggered by Until has the same effect as clicking the stop button.
- After the Until parameter is used, the normal transition is not triggered. If it is triggered, see the figure below.

Example of use:

	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions from the previous motion and the until parameter is met, the robot stops. After the robot stops, it is considered that both the previous and current movements have been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter is not transitioning and the until parameter is met, the robot stops. After the robot stops, it is considered that both the current movement has been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions to the next motion and the until parameter is met, it is considered that the until parameter is invalid. The robot runs normally and the until condition will not be triggered.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -65535 to +65535.	Movj/Movl/MovAbsJ/ Movc/Jump/JumpL

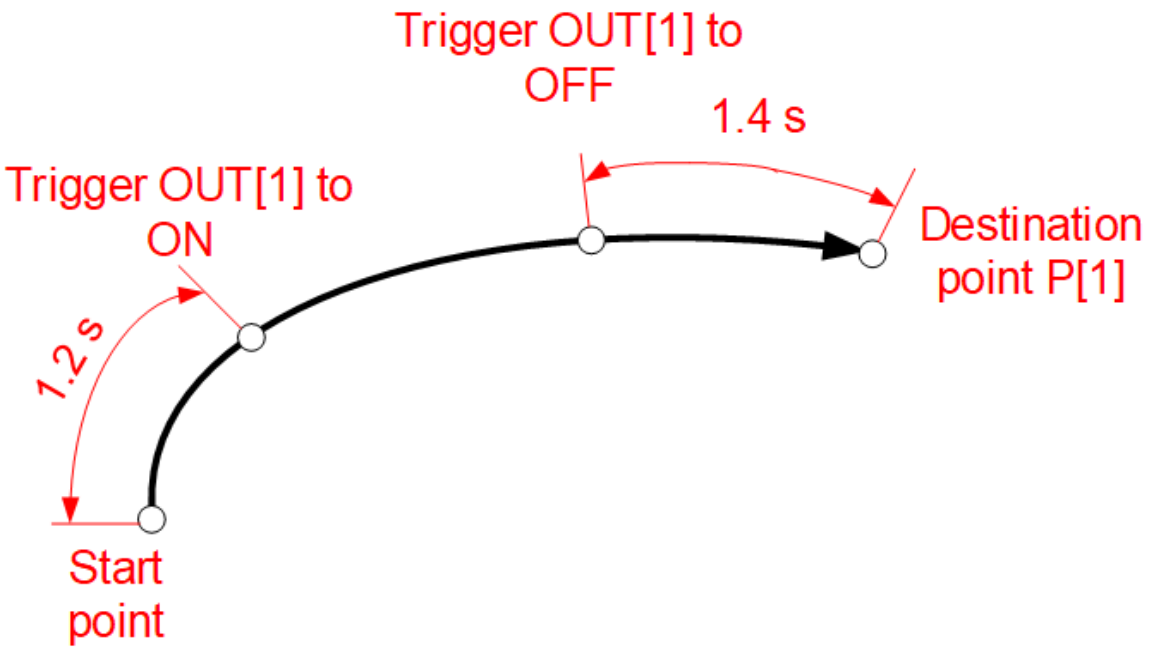
Type	Description	Applicable motion type
	n ≥ 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, the setting is invalid. In this case, the signal output is triggered at the start or end point of the motion at most.	
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when n% of the path is finished. The value ranges from -100.000 to +100.000. The negative number represents the percentage with the end point as the reference point.	Movj/Movl/MovAbsJ/ Movc/Jump/JumpL
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The value ranges from -65535 to +65535. n ≥ 0 indicates that signals are output after the robot moves by n mm from the start point. n < 0 indicates that signals are output before the robot moves to n mm from the end point.	Movl/Movc/JumpL

CAUTION

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- 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.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
- If you want the robot to pass through the singular position, you can only use Movj or MovAbsJ instruction.

- **Example:**

Movj P[1], V[30], Z[3], Tool[0], OUT(1,ON,T[1.2]), OUT(1,OFF,T[-1.4]);



SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.

7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.
9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:

- ◦ The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
  ◦ When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.
- **Example:**
  ◦ 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
  ◦ Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
  ◦ Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

```
SLDataClear All;
  For B[0]=0,B[0]<10,Step[1]
    Movj LP[0],V[30],Z[0],Tool[0],SLOff;
    Movj LP[1],V[30],Z[CP], Tool[0],SLOn;
    Movj LP[2],V[30],Z[CP], Tool[0],NWait,SLReset;
```

```
    Movj LP[3],V[30],Z[0], Tool[0];
    Set Out[1];
    SLVSMode MidLevel;
EndFor;
```

- PE description:

PE: Current point.

When the trajectory is cleared, the PE is repositioned to indicate the current position of the robot.

NOTE

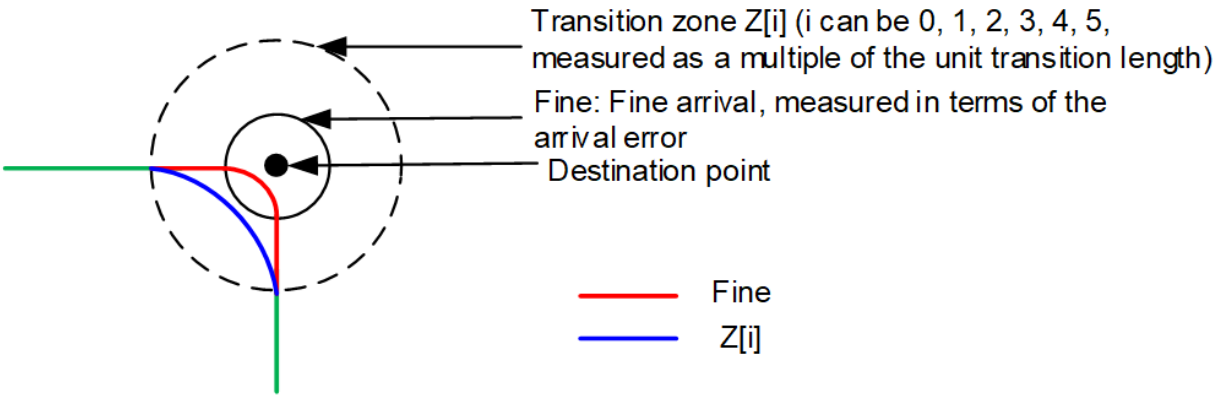
- For a motion instruction including PE, care should be taken if the previous motion instruction includes Until. In the case that the motion fails to complete since the previous motion instruction includes Until, if the program is not stopped and if the subsequent motion instruction uses Offset(PE,PR), then the point represented by PE is not the current point, but the target point in the previous motion instruction.
- If the previous motion instruction of PE is Home or ArmChange, then ArmChangePE becomes the target point of Home or ArmChange instruction.

- **Transition characteristics**:

The transition function is used to shorten the motion cycle time. The transition range is an area with the target point as the center and the transition length as the radius.

Transition length = Transition level x Transition unit length

Type	Description
Fine	It indicates precise arrival. The robot is considered to achieve precise arrival when it enters and stays within the set arrival error range. (See Arrival Error)
Z[0]	No transition. The robot cannot be ensured to reach the target point precisely.
Z[1]	Transition at 1x transition unit length.
Z[2]	Transition at 2x transition unit length.
Z[3]	Transition at 3x transition unit length.
Z[4]	Transition at 4x transition unit length.
Z[5]	Transition at 5x transition unit length.
Z[CP]	Transition at the maximum transition length (See Maximum Transition Length).



The unit transition length is divided into joint transition unit and linear transition unit. The instructions Movj/ MovAbsJ/Jump use joint transition unit (The J3 axis of SCARA robots still use linear transition unit), while other motion instructions use linear transition unit.

Restrictions on transition: In some cases, transition defined with Z[1] to Z[5] or Z[CP] is invalid and considered Z[0].

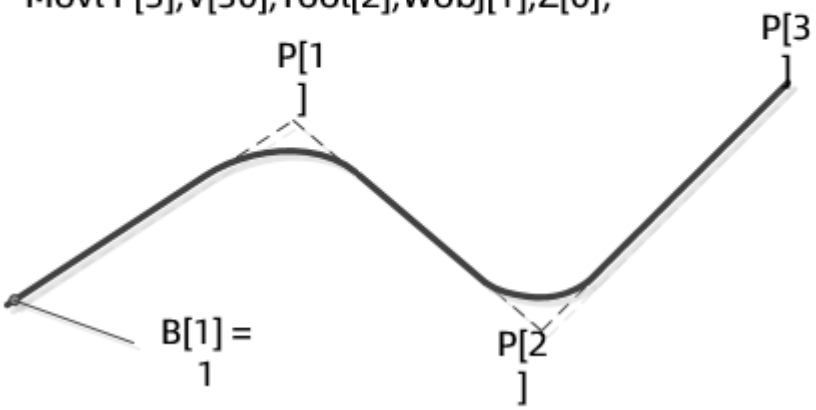
1. For a target point which is the end, the actual transition effect is always Z[0].
2. If a motion instruction uses the Until parameter, the actual transition effect is always Z[0].
3. If there are blocking non-motion instructions between motion instructions, the actual transition effect is always Z[0].
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.

> CAUTION
>
> 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 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

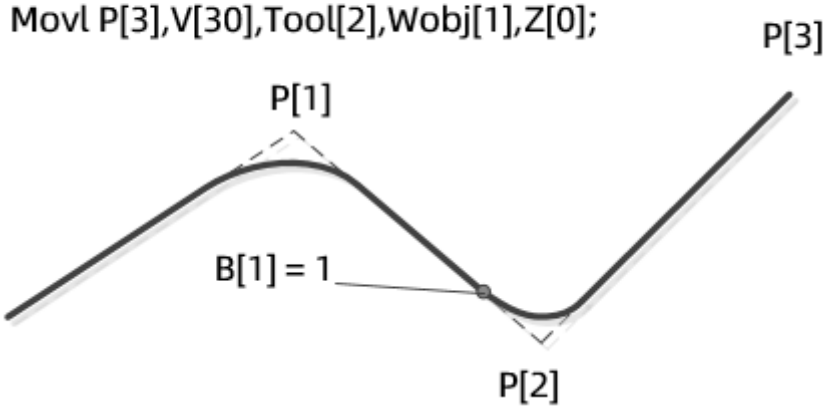
```
Movl P[1],V[30],Tool[2],Wobj[1],Z[3];
Movl P[2],V[30],Tool[2],Wobj[1],Z[3],NWAIT;
B[1] = 1;
Movl P[3],V[30],Tool[2],Wobj[1],Z[0];
```



length.

b. If the motion instruction does not include Nwait parameter, in the system default or when the transition wait feature 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.

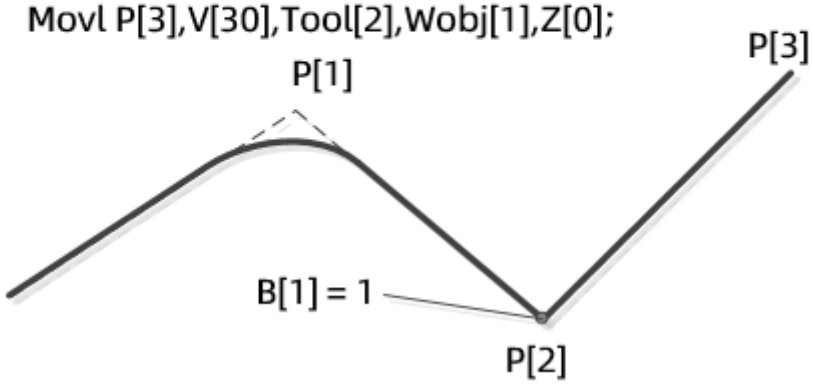
```
Movl P[1],V[30],Tool[2],Wobj[1],Z[3];
Movl P[2],V[30],Tool[2],Wobj[1],Z[3];
B[1] = 1;
Movl P[3],V[30],Tool[2],Wobj[1],Z[0];
```



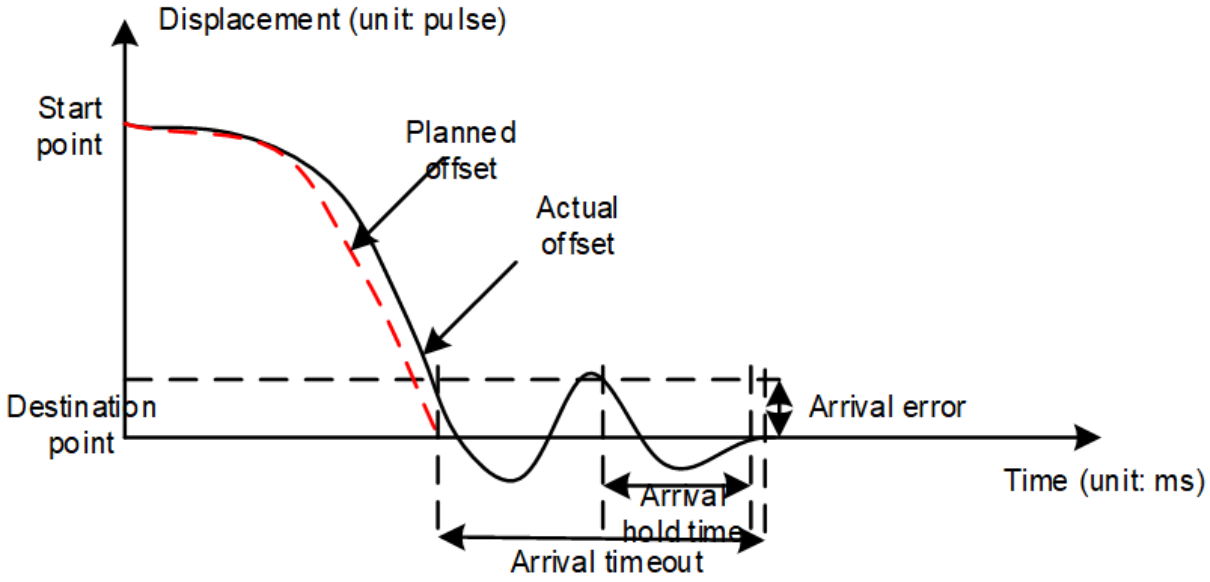
c. If the motion instruction does not include Nwait parameter, when the transition wait feature is enabled (SetFlyWait(ON)), the robot will slow down and stop at the target point before executing subsequent non-

motion instructions. In this case, the transition disappears.

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



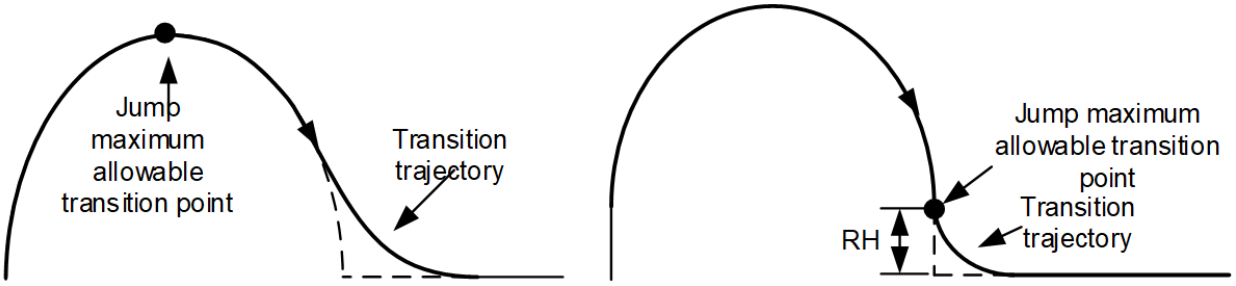
- **Arrival error:**



The arrival error includes:

- Arrival error: Default 80,000 pulses, which can be adjusted through settings in Manufacturer mode.
- Arrival hold time: Default 20 ms, which can be adjusted through settings in Manufacturer mode.
- Arrival timeout: System fixed value 2,000 ms.

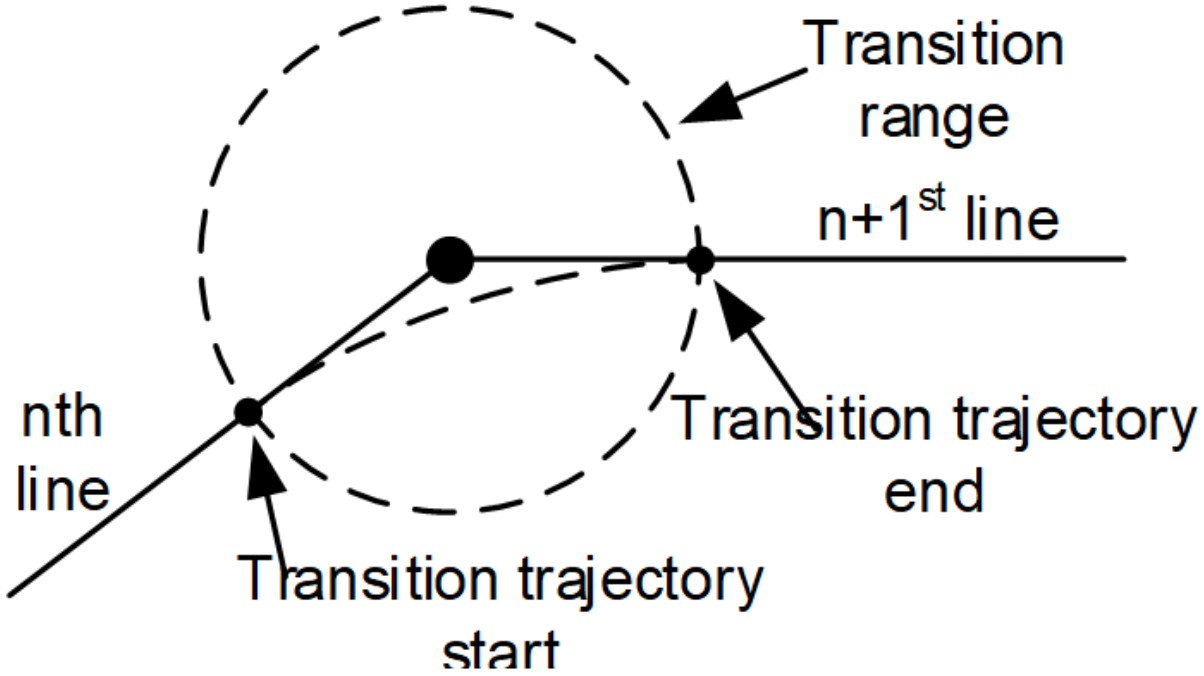
Maximum transition length: For specific motion, the allowable transition length is limited. When the set transition length exceeds the allowable maximum transition length, the actual transition length takes the maximum allowable transition length. The maximum allowable transition length of Movl, Movj, Movc, and MovAbsJ instructions is a half of the total length. As shown in the figure below, for Jump and JumpL instructions, when RH (or LH) is 0, the maximum allowable transition length is a half of the total length. When RH (or LH) is not 0, the maximum allowable transition length is RH (or LH).



Jump transition with RH=0

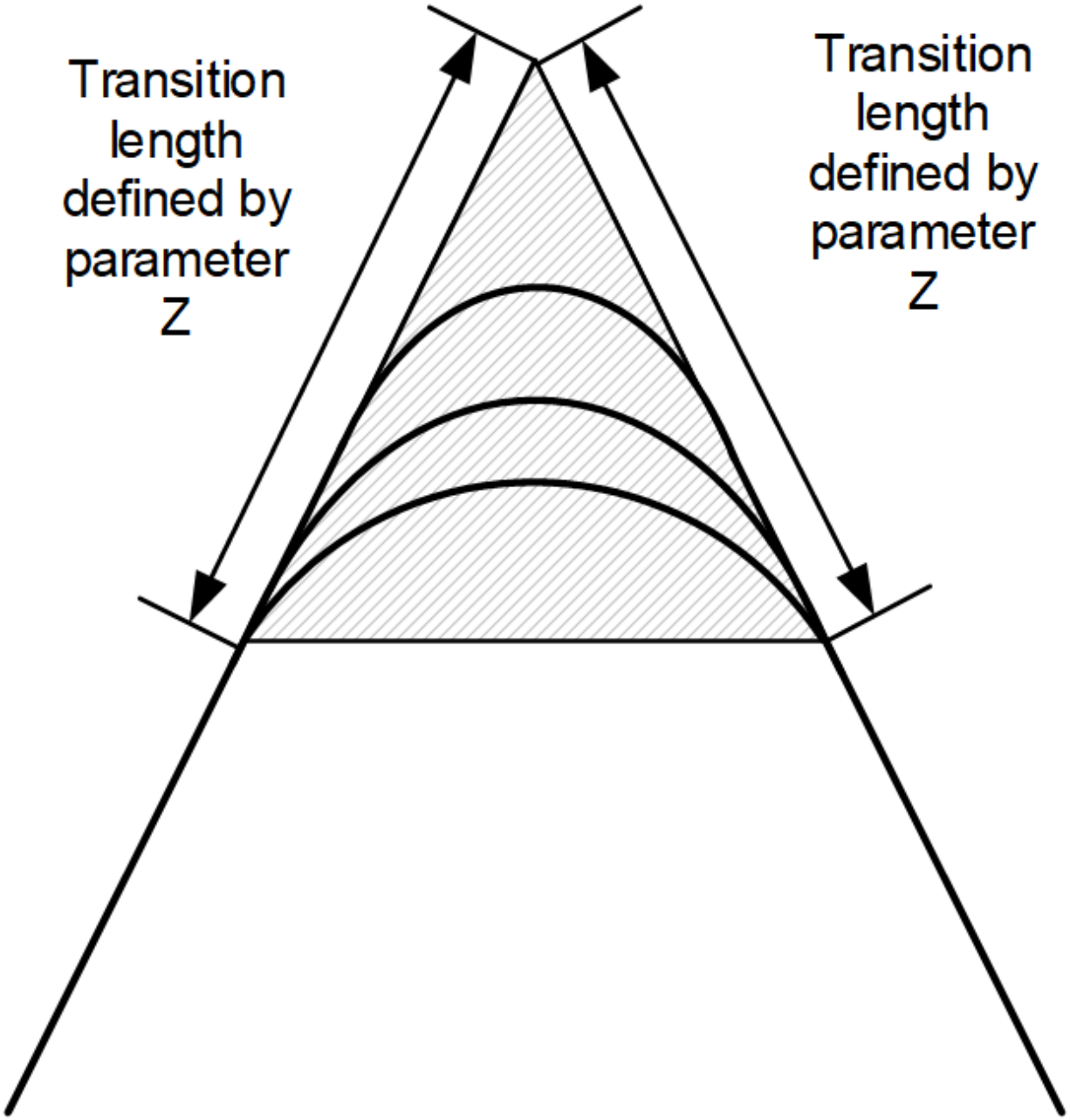
Jump transition with RH≠0

Impact of transition on motion I/O: When the dotted transition area shown in the following figure exists, the precision of motion I/Os in the transition area cannot be guaranteed. Actually, the precision take effect at an uncertain position between the start point and end point of the transition trajectory.



- **Cautions:**

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



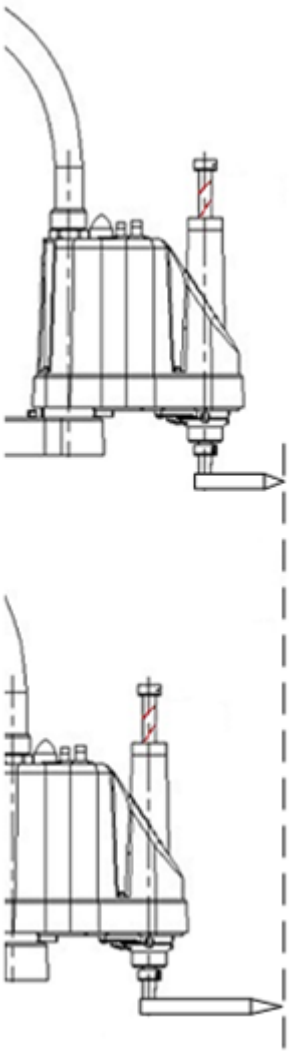
- **Extended use of Tool parameters**:

You can switch the tool used by using the Tool parameter in the motion instruction. Therefore, when a new tool is used, the new TCP can reach the position and orientation of the original TCP.

Application scenario: Vision-assisted workobject pickup requires calibration of the tool frame parameters using the camera. It is needed to dynamically change the tool frame parameters.

CAUTION

The tool change function only requires that the tool number in the position variable is selected correctly. The position variables can be taken under any frame, including joint and base frames.



- **Case:**

Previously, Tool1 was used for point teaching. The program is as follows.

Start;

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

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

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

...

End;

Now the tool is switched to Tool3. You can directly modify the tool parameter in the instruction, without having to perform calibration again.

Start;

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

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

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

...

End;

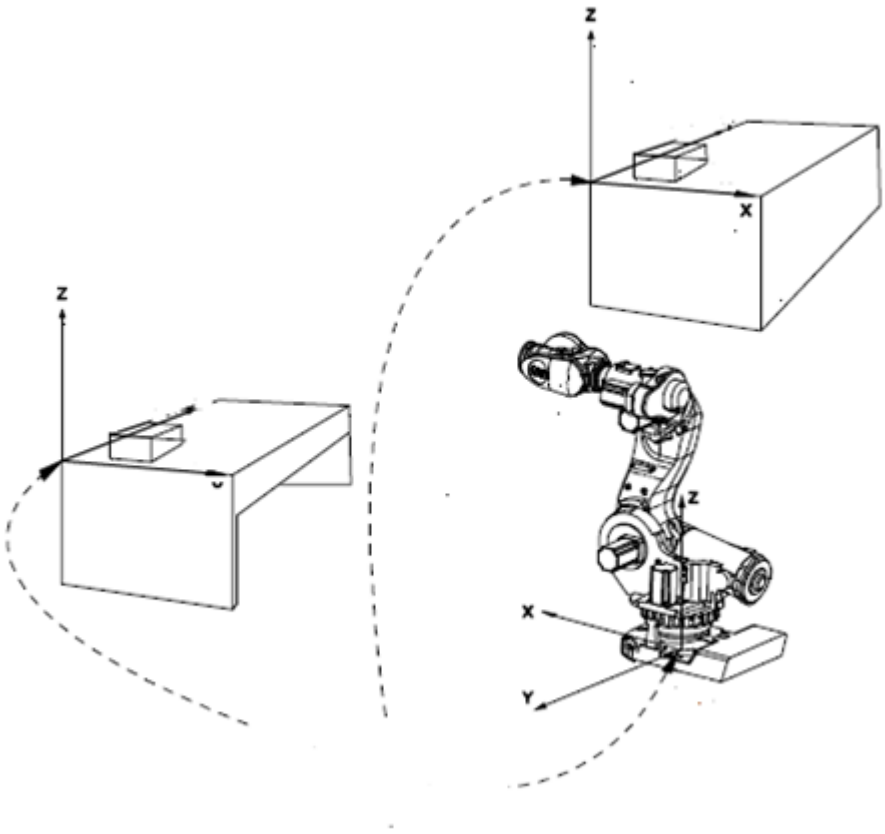
- **Extended use of Wobj parameters**:

You can switch the workobject frame by the Wobj parameter in the motion instruction.

Application scenario: It is used in applications with working points having the same relative positions on different workbenches.

CAUTION

Prerequisite: The points included in the original program must have been acquired under the workobject frame.



- **Case:**

Wobj1 frame was established on a workbench on which some work objects were placed. The Wobj1 frame was used as the current workobject frame. The robot was taught and programmed under this workobject frame.

Start;

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

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

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

...

End;

If the location of the workbench was changed. It is needed to establish a new Wobj2 frame. You can directly change as follows:

Start;

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

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

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

...

End;

- **Specifications of duplicate points:**

1. Two adjacent joints are considered as duplicate points if their difference in joint angle is smaller than 1e-5 radian.
2. For two motion instructions with duplicated points, the second motion instruction in the program is invalid, including motion dependent parameters such as motion I/Os.

4.2 Movl

- **Function**: Linear interpolation.
- **Description**: Moves the robot to a given target position.
- **Format**: Movl P, V, Z, Tool, [Wobj], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)…],[SLOn/SLOff/SLReset].
- **Parameter**:

P: Target point in Cartesian space, required.

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

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

PE can be used in Offset and OffsetT to indicate offset from the current position.

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

For details, see 9.5.7 Offset.

Pallet point getting form - Pallet(PalletNo, Row, Colunm, Lay), LPallet(PalletNo, Row, Colunm, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number. For details, see the 9.5.20 P=Pallet instruction.

V/Speed: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. Speed indicates the end linear speed. For details, see the 6.17 Speed instruction.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the maximum allowable transition length is equal to the full motion length of the instruction.

(2) If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.

(3) If the current instruction is Jump/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].

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length of the instruction to ensure that the two transition areas no longer overlap. If both transition areas at the beginning and end of the instruction are larger than half the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see Transition characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

In the tracking process, this parameter shall be replaced with Cnv. It is a mandatory parameter and is used to indicate motion in the tracking process.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

When not specified and Speed is used, Acc is 20% by default.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag.

Indicates to execute the subsequent non-motion instructions immediately before the target point is reached.

This causes subsequent non-motion instructions such as operation, signal and logic judgment to be executed in advance until the next motion instruction is encountered. Forced blocking will invalidate Nwait.

NOTE

- If the subsequent instruction is a motion instruction, regardless of whether this motion instruction includes NWait, the motion instruction will be issued in advance.
- In the case where multiple motion instructions are used, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance when the first motion instruction is executed, regardless of whether the first motion instruction includes NWait.
- The stop triggered by Until has the same effect as clicking the stop button.
- When the Until parameter is used, there is no transition between the starting and ending 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], Tool[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:**

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];NWait;

```
Set Out [6], ON
End;
```

Program 2:

```
Start;
Movl P [0], V[30], Z[0], Tool [0], Wobj[0];
Movl P [1], V[30], Z[0], Tool [0], Wobj[0];
Movl P [2], V[30], Z[0], Tool [0], Wobj[0];
Movl P [3], V[30], Z[0], Tool [0], Wobj[0];NWait;
Set Out [6], ON
Movl P [4], V[30], Z[0], Tool [0], Wobj[0];
Movl P [5], V[30], Z[0], Tool [0], Wobj[0];
Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;
Set Out [7], ON
End;
```

Program 3:

```
Start;
Movl P [0], V[30], Z[0], Tool [0], Wobj[0];
Movl P [1], V[30], Z[0], Tool [0], Wobj[0];
Movl P [2], V[30], Z[0], Tool [0], Wobj[0];
Movl P [3], V[30], Z[0], Tool [0], Wobj[0];
Set Out [6], ON
Movl P [4], V[30], Z[0], Tool [0], Wobj[0];
Movl P [5], V[30], Z[0], Tool [0], Wobj[0];
Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;
Set Out [7], ON
End;
```

In Program 1, 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.

In Program 2, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

In Program 3, Set Out[7] will be executed in advance, and in fact, both Set Out[6] and Set Out[7] are executed at the fifth 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 to the end.

- **Subparameter:**

In: Input signal, only supports In[0]-In[127].

Value: Input signal value, ON or OFF

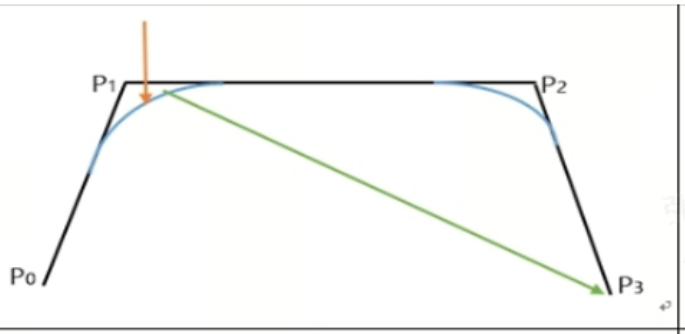
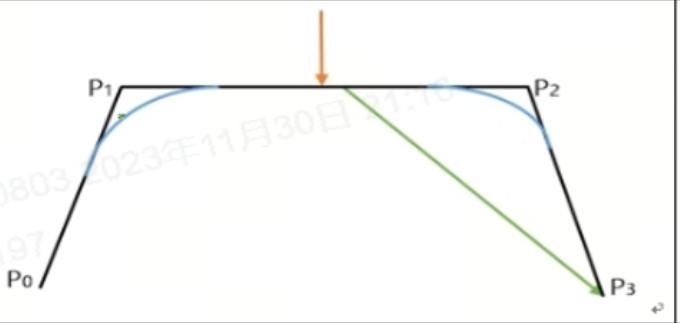
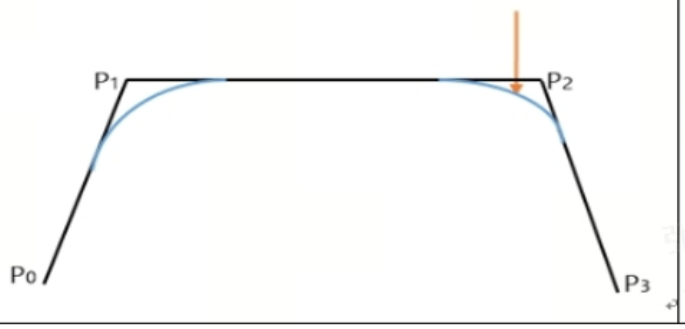
Example of use:

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

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

CAUTION

- If the Until parameter is used, Z is considered Z[0].
- When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will be executed in advance.
- The stop triggered by Until has the same effect as clicking the stop button.
- After the Until parameter is used, the normal transition is not triggered. If it is triggered, see the figure below.
  Example of use:

	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions from the previous motion and the until parameter is met, the robot stops. After the robot stops, it is considered that both the previous and current movements have been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter is not transitioning and the until parameter is met, the robot stops. After the robot stops, it is considered that both the current movement has been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions to the next motion and the until parameter is met, it is considered that the until parameter is invalid. The robot runs normally and the until condition will not be triggered.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -65535 to +65535.	Movj/Movl/MovAbsJ/Movc /Jump/JumpL

Type	Description	Applicable motion type
	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, the setting is invalid. In this case, the signal output is triggered at the start or end point of the motion at most.	
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when n% of the path is finished. The value ranges from -100.000 to +100.000. The negative number represents the percentage with the end point as the reference point.	Movj/Movl/MovAbsJ/Movc /Jump/JumpL
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The value ranges from -65535 to +65535. n ≥ 0 indicates that signals are output after the robot moves by n mm from the start point. n < 0 indicates that signals are output before the robot moves to n mm from the end point.	Movl/Movc/JumpL

CAUTION

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- When T[n]/Ds[n]/S[n] is set beyond the motion range, the signal output is triggered at the start or end point of the motion at most.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
- When CP motion instructions (Movl/Movc/JumpL/Movs) are executed near the axis limit of a joint, false alarms of triggering axis limit in the planned trajectory may occur. This is a specification related issue, and the recommended actions include avoiding the axis limit or using PTP motion instructions (Movj/ MovAbsJ/Jump) instead.

- **Example of use:**

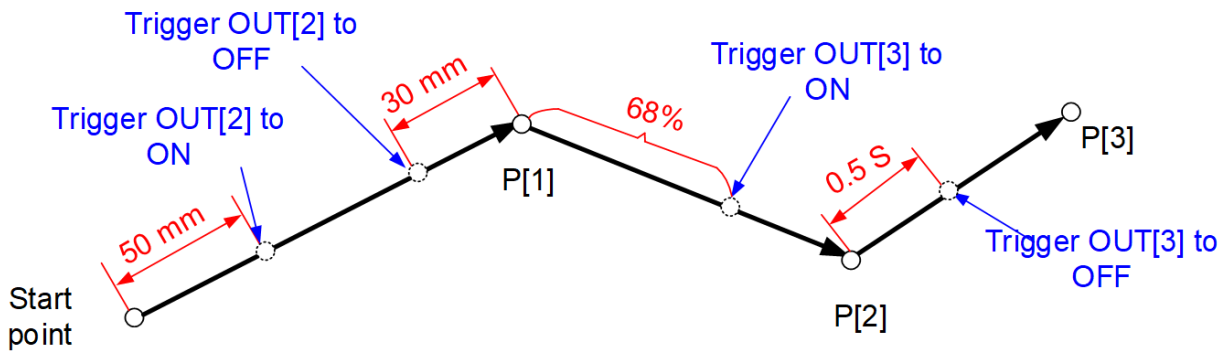
START;

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

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

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

END;



SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.
7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn

parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.

9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:

- ◦ The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
  ◦ When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.
- **Example:**
  ◦ 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
  ◦ Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
  ◦ Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

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

4.3 MovAbsJ

- **Description**: Quickly moves the robot from one joint point to another joint point. The trajectory is usually not a straight line, and all axes reach the target positions simultaneously.
- **Format**: MovAbsJ JP, V, Z, Tool, [Wobj], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)…], [SLOn/SLOff/SLReset].
- **Parameter**:

JP: Target point (joint value) to reach, required.

Common form: JP[1].

V: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. The Speed instruction is not supported.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the transition length is equal to the full motion length of the instruction.

(2) If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.

(3) If the current instruction is Jump/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].

CAUTION

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length 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 the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For more information, see Transition Characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag.

Indicates to execute the subsequent non-motion instructions immediately before the target point is reached.

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

NOTE

- If the subsequent instruction is a motion instruction, regardless of whether this motion instruction includes NWait, the motion instruction will be issued in advance.
- In the case where multiple motion instructions are used, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance when the first motion instruction is executed, regardless of whether the first motion instruction includes NWait.

- **Example of use:**

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

MovAbsJ JP[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:**

Program 1:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [6], ON

End;

Program 2:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Set Out [6], ON

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];

Movl P [5], V[30], Z[0], Tool [0], Wobj[0];

Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [7], ON

End;

Program 3:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Set Out [6], ON

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];

Movl P [5], V[30], Z[0], Tool [0], Wobj[0];

Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [7], ON

End;

In Program 1, 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.

In Program 2, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

In Program 3, Set Out[7] will be executed in advance, and in fact, both Set Out[6] and Set Out[7] are executed at the fifth 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 to the end.

- **Subparameter:**

In: Input signal, only supports In[0]-In[127].

Value: Input signal value, ON or OFF

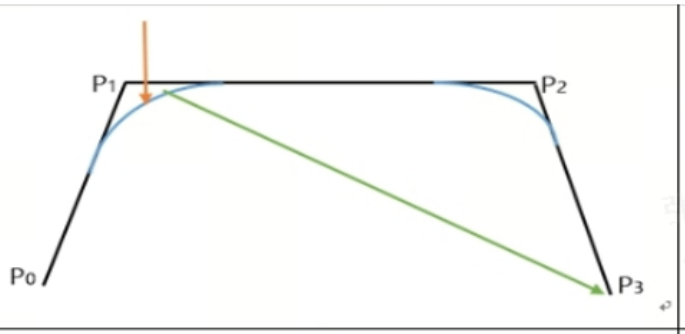
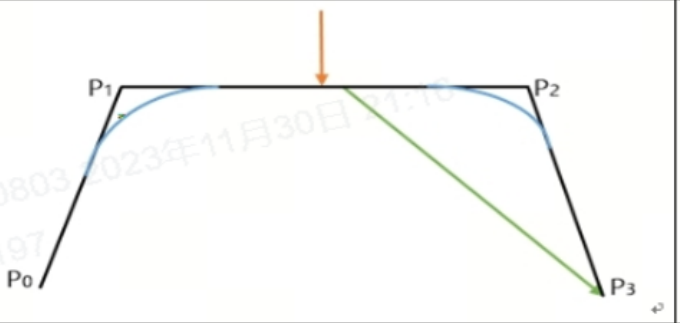
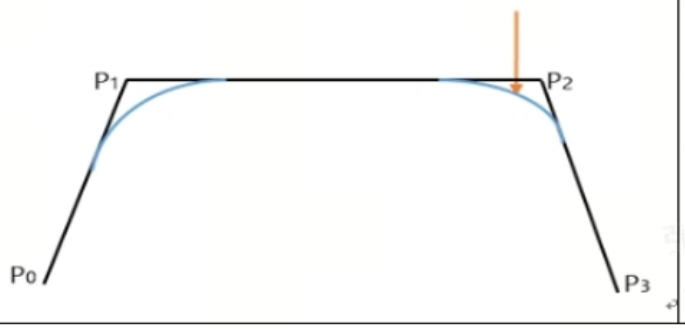
Example of use:

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

MovAbsJ JP[1],V[30],Z[0], Tool[0], Until In[5] == ON;

CAUTION

- If the Until parameter is used, Z is considered Z[0].
- When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will be executed in advance.
- The stop triggered by Until has the same effect as clicking the stop button.
- After the Until parameter is used, the normal transition is not triggered. If it is triggered, see the figure below.
  Example of use:

	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions from the previous motion and the until parameter is met, the robot stops. After the robot stops, it is considered that both the previous and current movements have been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter is not transitioning and the until parameter is met, the robot stops. After the robot stops, it is considered that both the current movement has been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions to the next motion and the until parameter is met, it is considered that the until parameter is invalid. The robot runs normally and the until condition will not be triggered.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -65535 to +65535.	Movj/Movl/MovAbsJ/Movc /Jump/JumpL

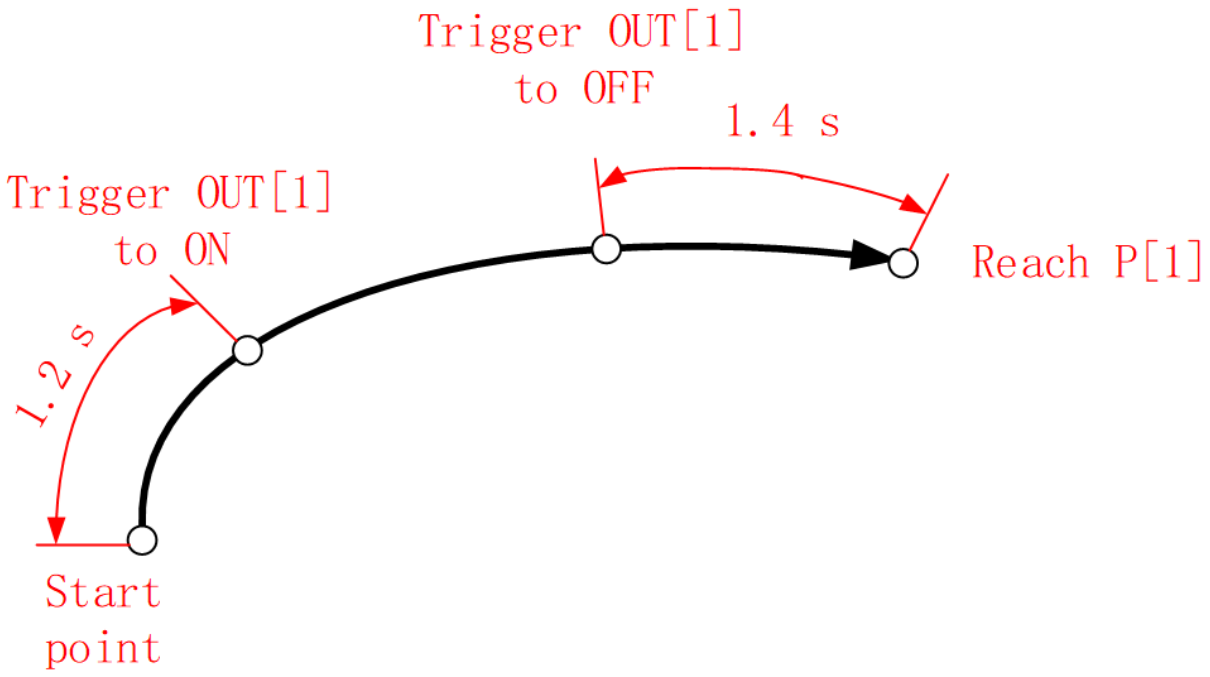
Type	Description	Applicable motion type
	n ≥ 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point. When time is set beyond the motion range, the setting is invalid. In this case, the signal output is triggered at the start or end point of the motion at most.	
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when n% of the path is finished. The value ranges from -100.000 to +100.000. The negative number represents the percentage with the end point as the reference point.	Movj/Movl/MovAbsJ/Movc /Jump/JumpL
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The value ranges from -65535 to +65535. n ≥ 0 indicates that signals are output after the robot moves by n mm from the start point. n < 0 indicates that signals are output before the robot moves to n mm from the end point.	Movl/Movc/JumpL

CAUTION

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- 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.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
- If you want the robot to pass through the singular position, you can only use Movj or MovAbsJ instruction.

Example:

MovAbsJ JP[1] ,V[30], Z[3], Tool[0],OUT(1,ON,T[1.2]), OUT(1,OFF,T[-1.4]);



SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.

7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.
9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:

- ◦ The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
  ◦ Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
  ◦ When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.
- **Example:**
  ◦ 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
  ◦ Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
  ◦ Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

```
SLDataClear All;
  For B[0]=0,B[0]<10,Step[1]
    MovAbsJ JP[0],V[30],Z[0],Tool[0],SLOff;
    MovAbsJ JP[1],V[30],Z[CP], Tool[0],SLOn;
    MovAbsJ JP[2],V[30],Z[CP], Tool[0],NWait,SLReset;
```

```
    MovAbsJ JP[3],V[30],Z[0], Tool[0];
    Set Out[1];
    SLVSMode MidLevel;
EndFor;
```

4.4 Movc

- **Function**: Circular interpolation.
- **Description**: Moves the robot to a given target position by circular motion.
- **Format**: Movc P, V, Z, Tool, [Wobj], [Acc], [Dec], [NWait], [Until In == value], [Out(No,value,Type)…],[SLOn/SLOff/SLReset].
- **Parameter**:

P: Target point in Cartesian space, required.

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

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

PE can be used in Offset and OffsetT to indicate offset from the current position.

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

For details, see 9.5.7 Offset.

Pallet point getting form: Pallet(PalletNo, Row, Colunm, Lay), LPallet(PalletNo, Row, Colunm, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number. For details, see the 9.5.20 P=Pallet instruction.

V/Speed: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. Speed indicates the end linear speed. For details, see the 6.17 Speed instruction.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the maximum allowable transition length is equal to the full motion length of the instruction.

(2) If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.

(3) If the current instruction is Jump/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].

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length of the instruction to ensure that the two transition areas no longer overlap. If both transition areas at the beginning and end of the instruction are larger than half the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see Transition characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

In the tracking process, this parameter shall be replaced with Cnv. It is a mandatory parameter and is used to indicate motion in the tracking process.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

When not specified and Speed is used, Acc is 20% by default.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag.

Indicates to execute the subsequent non-motion instructions immediately before the target point is reached.

This causes subsequent non-motion instructions such as operation, signal and logic judgment to be executed in advance until the next motion instruction is encountered. Forced blocking will invalidate Nwait.

NOTE

- If the subsequent instruction is a motion instruction, regardless of whether this motion instruction includes NWait, the motion instruction will be issued in advance.
- In the case where multiple motion instructions are used, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance when the first motion instruction is executed, regardless of whether the first motion instruction includes NWait.

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:**

Program 1:

```
Start;
    Movl P [0], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [1], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [2], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [3], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [4], V[30], Z[0], Tool [0], Wobj[0];NWait;
    Set Out [6], ON
End;
```

Program 2:

```
Start;
    Movl P [0], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [1], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [2], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [3], V[30], Z[0], Tool [0], Wobj[0];NWait;
    Set Out [6], ON
    Movl P [4], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [5], V[30], Z[0], Tool [0], Wobj[0];
    Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;
    Set Out [7], ON
```

End;

Program 3:

Start;

Movl P [0], V[30], Z[0], Tool [0], Wobj[0];

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

Movl P [2], V[30], Z[0], Tool [0], Wobj[0];

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

Set Out [6], ON

Movl P [4], V[30], Z[0], Tool [0], Wobj[0];

Movl P [5], V[30], Z[0], Tool [0], Wobj[0];

Movl P [6], V[30], Z[0], Tool [0], Wobj[0];NWait;

Set Out [7], ON

End;

In Program 1, 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.

In Program 2, both Set Out[6] and Set Out[7] will be executed in advance, and in fact, they are immediately executed following the first statement.

In Program 3, Set Out[7] will be executed in advance, and in fact, both Set Out[6] and Set Out[7] are executed at the fifth 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 to the end.

- **Subparameter:**

In: Input signal, only supports In[0]-In[127].

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 to the end.

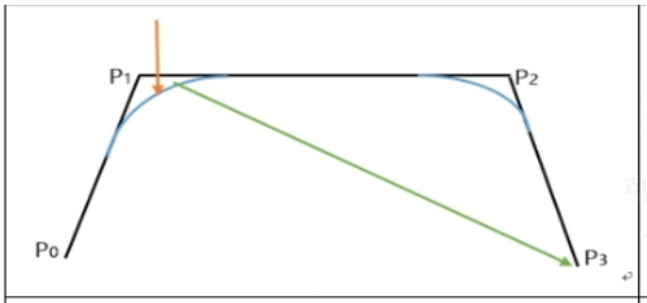
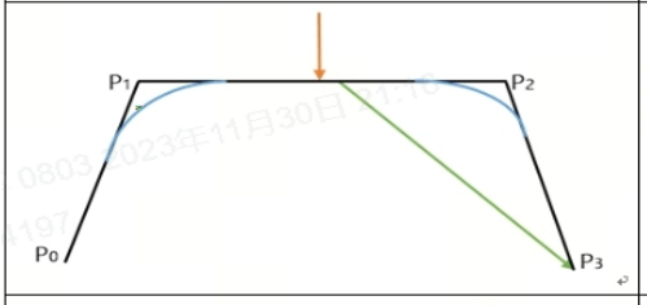
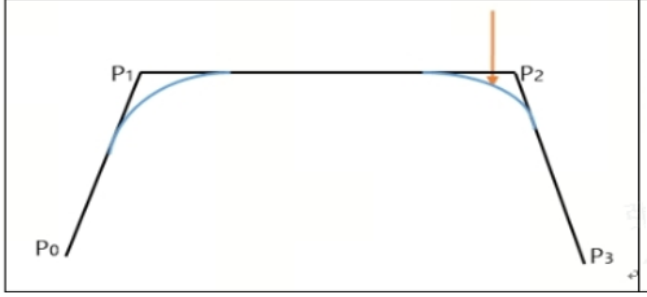
Movc P[1],V[30],Z[0],Tool[0],Wobj[0],Until In[5] == ON;

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

CAUTION

- If the Until parameter is used, Z is considered Z[0].
- When the instruction includes both the Until and NWait parameters, the subsequent non-motion instructions will be executed in advance.
- The stop triggered by Until has the same effect as clicking the stop button.
- After the Until parameter is used, the normal transition is not triggered. If it is triggered, see the figure below.

Example of use:

	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions from the previous motion and the until parameter is met, the robot stops. After the robot stops, it is considered that both the previous and current movements have been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter is not transitioning and the until parameter is met, the robot stops. After the robot stops, it is considered that both the current movement has been completed. Then the robot moves from P3 the stop point.
	Program is writen as follows: movl P1; movl P2 until; movl P3; If the instruction with the until parameter transitions to the next motion and the until parameter is met, it is considered that the until parameter is invalid. The robot runs normally and the until condition will not be triggered.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -65535 to +65535.	Movj/Movl/MovAbsJ/Movc/Jump/JumpL

Type	Description	Applicable motion type
	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, the setting is invalid. In this case, the signal output is triggered at the start or end point of the motion at most.	
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when n% of the path is finished. The value ranges from -100.000 to +100.000. The negative number represents the percentage with the end point as the reference point.	Movj/Movl/MovAbsJ/Movc/Jump/JumpL
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The value ranges from -65535 to +65535. n ≥ 0 indicates that signals are output after the robot moves by n mm from the start point. n < 0 indicates that signals are output before the robot moves to n mm from the end point.	Movl/Movc/JumpL

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- When T[n]/Ds[n]/S[n] is set beyond the motion range, the signal output is triggered at the start or end point of the motion at most.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
- When CP motion instructions (Movl/Movc/JumpL/Movs) are executed near the axis limit of a joint, false alarms of triggering axis limit in the planned trajectory may occur. This is a specification related issue, and the recommended actions include avoiding the axis limit or using PTP motion instructions (Movj/MovAbsJ/Jump) instead.

SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.

- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.
7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.
9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:
    - The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.

- Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
- When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.

Example:

- 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
- Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
- Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

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

CAUTION

- Three points determine an arc, so the instruction Movc must appear in pairs and cannot be separated by any instructions. The instruction before the Movc instruction must be a motion instruction (Movj, Movl, Movc, Jump, JumpL). Otherwise, an error "Lack of motion instruction before arc motion" occurs. These requirements must be followed. Even a comment can affect the results and cause alarms.
- If a circular motion includes two Movc instructions, all parameters except P are subject to the second one. At the same time, the first and second Movc instructions must be consistent in the tool number and wobj number.
- Do not use Movc in combination with PE as it may cause exceptions. Typical example: Movc Offset(PE,PR0)……
- For a Movc instruction that appears for the first time in the program, the motion instruction before it must not be used in combination with PE. For example:

Correct program:

```
Start;
    Movl P[0] ,V[30] ,Z[0],Tool[1],Wobj[2];
    Movc P[1] ,V[30] ,Z[0],Tool[1],Wobj[2];
    Movc P[2] ,V[30] ,Z[0],Tool[1],Wobj[2];
End;
```

Incorrect program:

```
Start;
    Movl Offset(PE,PR[0]),P[0],Z[0],Tool[1],Wobj[2];
    Movc P[1] ,V[30] ,Z[0],Tool[1],Wobj[2];
    Movc P[2] ,V[30] ,Z[0],Tool[1],Wobj[2];
End;
```

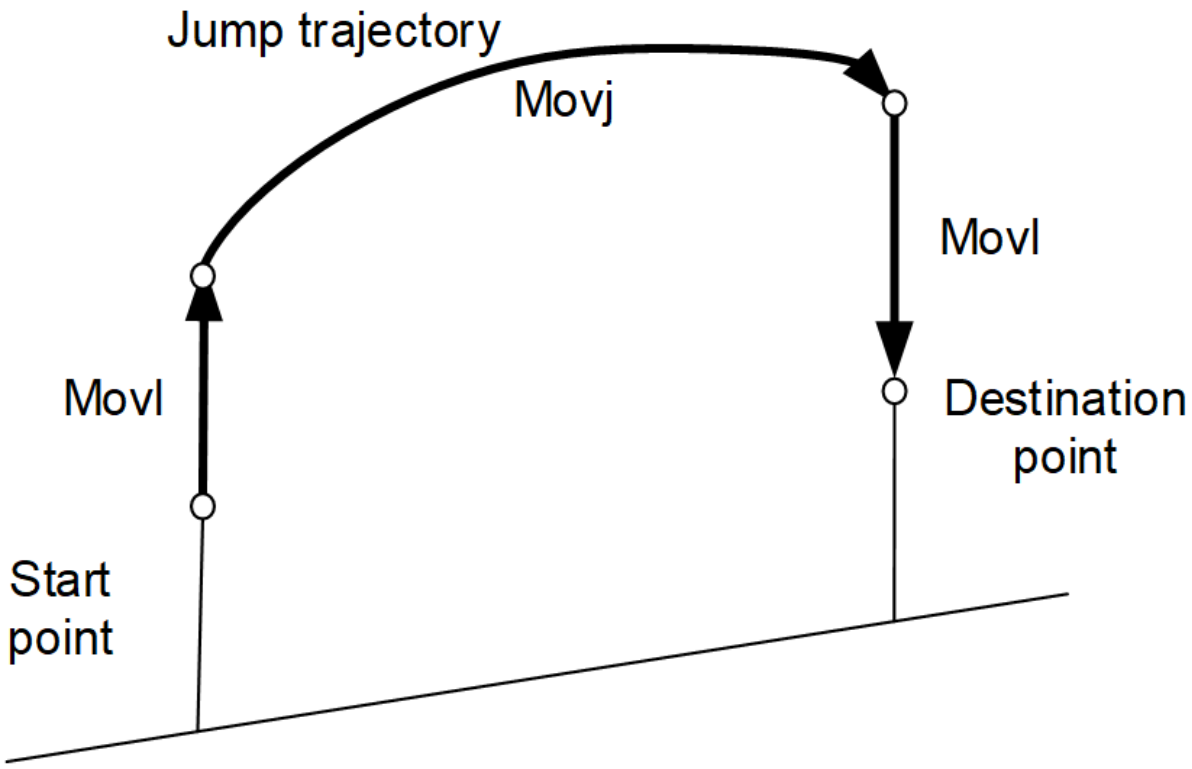
- When not used in combination with PE, it can be used in the program for auto running and single-step teaching. Even if you pause the program halfway, the robot can continue to move to the end of the arc. Example:

```
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 reach P[3] and the trajectory is circular.
Movc    P[3] ,V[80] ,Z[3];
END;
```

- The first Movc instruction only provides the data of the transition point. The overall motion is only based on the parameters in the second Movc instruction.
- During motion with the Movc instruction, in the default mode, only the position of the intermediate point is passed, but not necessarily the orientation of the intermediate point. By using the SetCirOriMode instruction, the robot can synchronously reach both the position and orientation of the intermediate point during circular motion.

4.5 Jump

- **Function**: Jump instruction.
- **Description**: The motion process is divided into three segments: the ascending segment Movl, the middle segment Movj and the descending segment Movl.



- **Format**: Jump P, V, Z, Tool, [Wobj], [Acc], [Dec] [NWait], [Out(No, value, Type)…], LH, MH, RH, [SLOn/SLOff/SLReset]
- **Parameter**:

P: Target point in Cartesian space, required.

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

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

PE can be used in Offset and OffsetT to indicate offset from the current position.

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

For details, see 9.5.7 Offset.

Pallet point getting form: Pallet(PalletNo, Row, Colunm, Lay), LPallet(PalletNo, Row, Colunm, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number. For details, see the 9.5.20 P=Pallet instruction.

V: Specifies a speed. For example, V[50] represents 50% of the maximum speed. The Speed instruction is not supported.

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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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.

- If the current instruction is Movj/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the maximum allowable transition length is equal to the full motion length of the instruction.
- If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.
- If the current instruction is Jump/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].

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length of the instruction to ensure that the two transition areas no longer overlap. If both transition areas at the beginning and end of the instruction are larger than half the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see Transition characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag. For details, see the NWait parameter of the Movj instruction.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

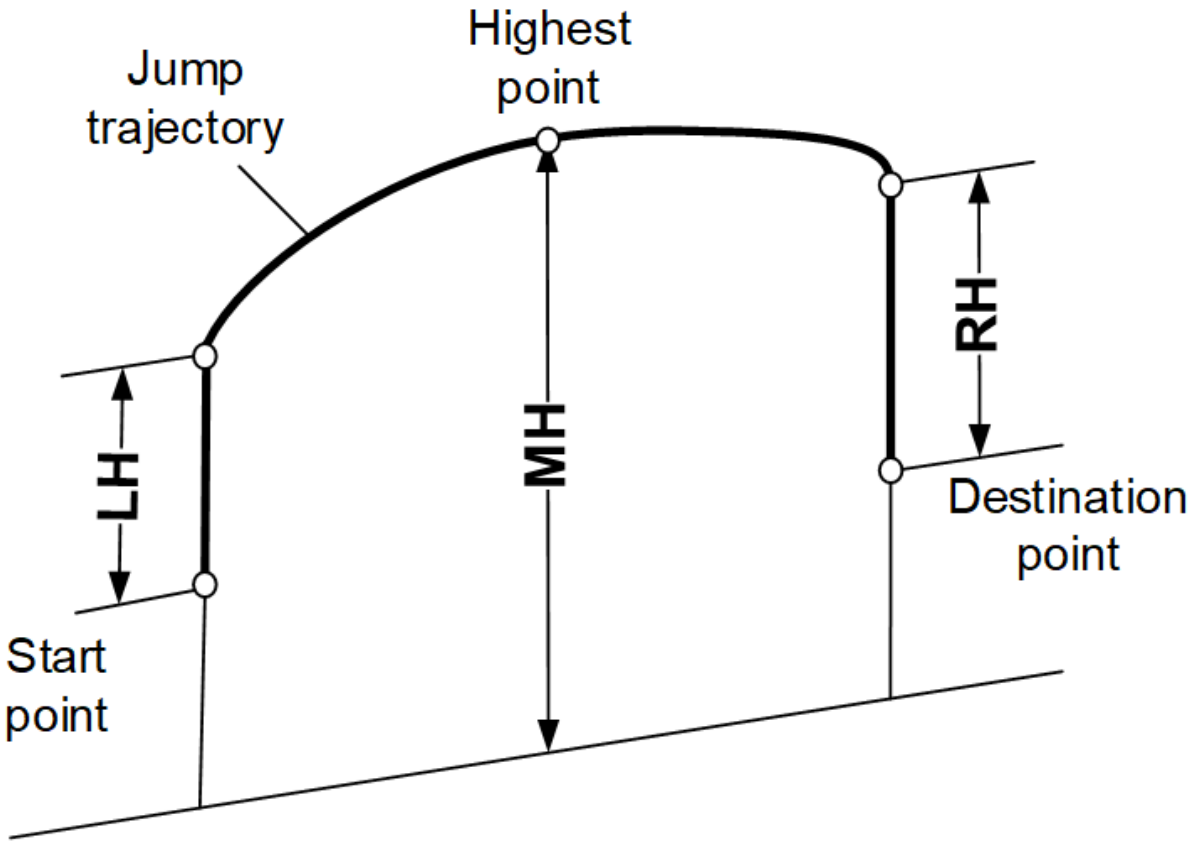
Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -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, the setting is invalid. In this case, the signal output is triggered at the start or end point of the motion at most.	Movj/Movl/MovAbsJ/ Movc/Jump/JumpL
Ds[n]	Signals are output by a path percentage in the motion process. Ds[n] indicates a path percentage. A signal is output when n% of the path is finished. The value ranges from -100.000 to +100.000. The negative number represents the percentage with the end point as the reference point.	Movj/Movl/MovAbsJ/ Movc/Jump/JumpL
S[n]	Signals are output by distance in the motion process. S[n] indicates distance in mm. The value ranges from -65535 to +65535. n ≥ 0 indicates that signals are output after the robot moves by n mm from the start point. n < 0 indicates that signals are output before the robot moves to n mm from the end point.	Movl/Movc/JumpL

CAUTION

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- When T[n]/Ds[n]/S[n] is set beyond the motion range, the signal output is triggered at the start or end point of the motion at most.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
  See the figure below:



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

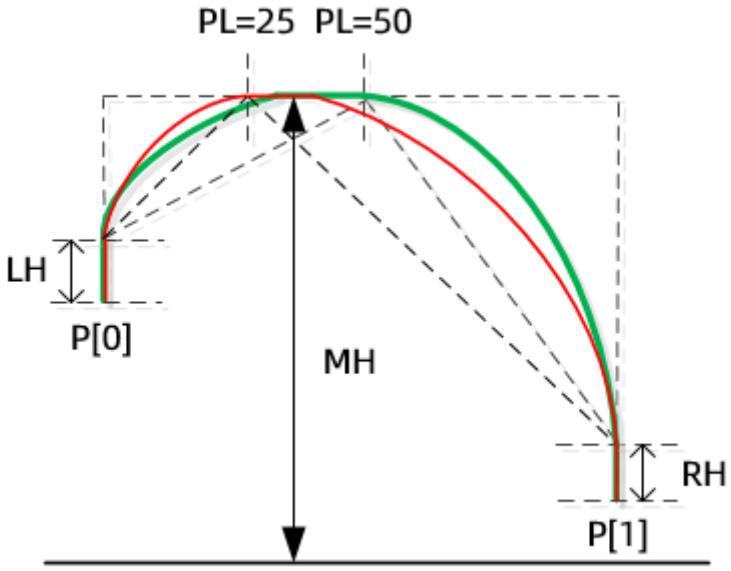
RH: Vertical motion height at the destination point position; format: RH[Z]; value range 0.000 to 2000.000.

MH: Total height of motion. Format:

- MH[Z], value range -2000.000 to +2000.000
- 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.

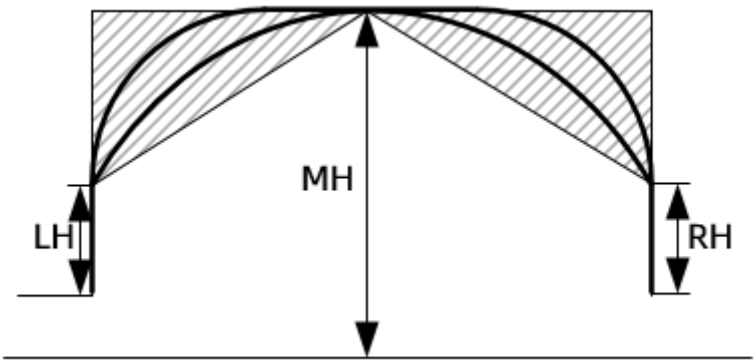
- Parameter type: Integer.
- Range: 20 to 80.
- Default value: 50 (horizontal midpoint).



CAUTION

- The PL parameter is only valid for the current Jump motion. If you need to adjust the highest point for multiple Jump instructions, set the additional parameters of the corresponding instructions.
- 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, check whether the robot path interferes with the surrounding objects at low speed first. For the initial setup, follow the default value 50 for trial run and gradually adjust it according to the Z-axis displacement ratio on both sides of Jump to avoid the robot path interfering with the surrounding environment.
- When the proportion of the highest point is set too large or too small, the robot's motion efficiency may be reduced. Therefore, be careful to adjust the highest point position reasonably.
- 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. The trajectory formed is controlled within the shaded triangle. However, it is not entirely certain and may change with parameters such as speed and acceleration. It is important to confirm the motion trajectory at the actual operating speed to avoid collisions.



SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.
7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.

9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:
    - The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
    - When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.

11. For a Jump instruction with SLOn/SLReset, by default the robot automatically determines whether the Jump instruction is suitable for segmentwise self-learning based on the motion parameters of the instruction:
    - The robot necessarily stops at the end of the third (descending) segment where it performs self-learning. In addition to this, the robot determines whether additional self-learning is needed at the end of the first (ascending) segment and second (horizontal) segment of the Jump instruction.
    - If segmentwise self-learning is not needed, you can set the self-learning mode of Jump motion through the SetJumpSLMode instruction. For details, see the SetJumpSLMode instruction.

- **Example:**
    - 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
    - Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
    - Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

```
SLDataClear All;
  For B[0]=0,B[0]<10,Step[1]
    Movj LP[0],V[30],Z[0],Tool[0],Wobj[0],SLOff;
    Movj LP[1],V[30],Z[CP],Tool[0],Wobj[0],SLOn;
```

Movj LP[2],V[30],Z[CP],Tool[0],Wobj[0],NWait,SLReset;

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

Set Out[1];

SLVSMode MidLevel;

EndFor;

CAUTION

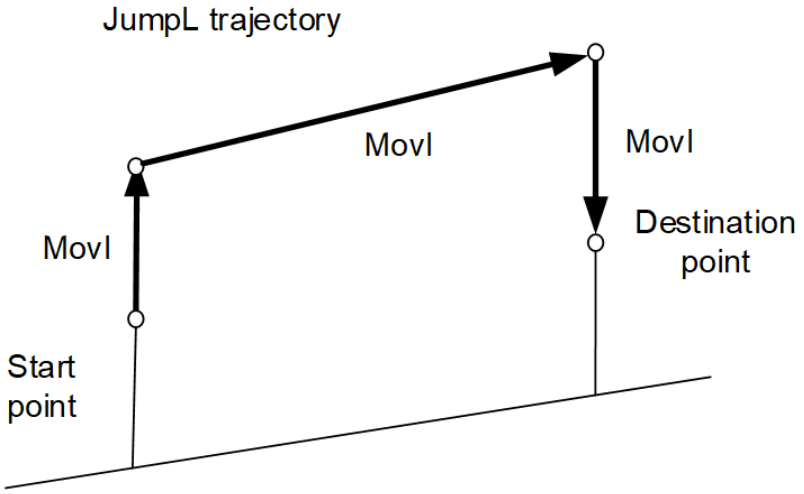
- The Jump instruction applies to only the SCARA and Delta robots.
- Because the zero point of the base frame of the SCARA robot is above, MH is negative in most cases.
- Generally, the height parameter must satisfy MH > start point height + LH and MH > end point height + RH.
- For any settings that do not satisfy these conditions, a non-gate shaped motion may occur, which can be used for special applications.

Example:

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

4.6 JumpL

- **Function**: Linear jump instruction, with the trajectory of intermediate segment being a straight line.
- **Description**: The motion process is divided into three segments: the ascending segment Movl, the middle segment Movl and the descending segment Movl.



- **Format**: JumpL P, V, Z, Tool, [Wobj], [Acc], [Dec] [NWait], [Out(No, value, Type)…], LH, MH, RH, [SLOn/SLOff/SLReset]
- **Parameter**:

P: Target point in Cartesian space, required.

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

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

PE can be used in Offset and OffsetT to indicate offset from the current position.

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

For details, see 9.5.7 Offset.

Pallet point getting form - Pallet(PalletNo, Row, Colunm, Lay), LPallet(PalletNo, Row, Colunm, Lay). Get the points on the pallet based on the pallet number, row number, column number, and layer number. For details, see the 9.5.20 P=Pallet instruction.

V/Speed: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. Speed indicates the end linear speed. For details, see the 6.17 Speed instruction.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the maximum allowable transition length is equal to the full motion length of the instruction.

(2) If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.

(3) If the current instruction is Jump/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].

CAUTION

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length of the instruction to ensure that the two transition areas no longer overlap. If both transition areas at the beginning and end of the instruction are larger than half the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.

For details, see Transition characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

In the tracking process, this parameter shall be replaced with Cnv. It is a mandatory parameter and is used to indicate motion in the tracking process.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag. For details, see the Movj instruction.

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

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

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

Type: Type of output signal in the motion process. The Type varies depending on the type of motion.

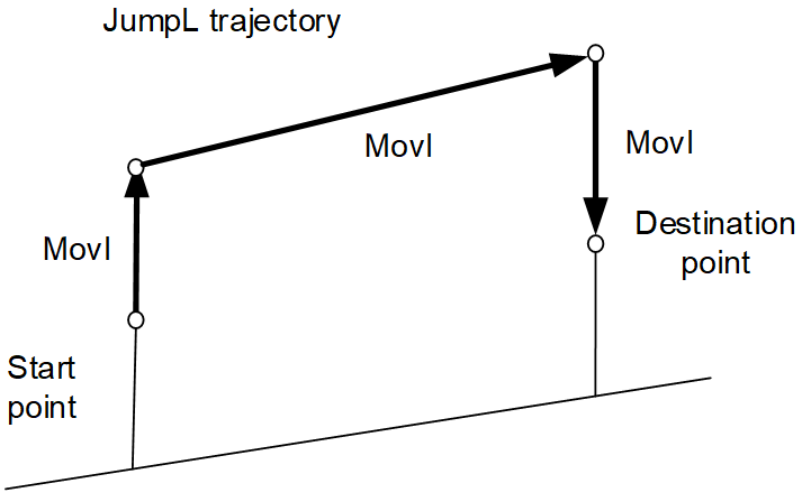
Type	Description	Applicable motion type
T[n]	Signals are output by time in the motion process. T[n] is time in seconds. The range is -65535 to +65535. n ≥ 0 indicates that signals are output n seconds after motion starts. n < 0 indicates that signals are output n seconds before arrival at a target point.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

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- When T[n]/Ds[n]/S[n] is set beyond the motion range, the signal output is triggered at the start or end point of the motion at most.
- When Jump and JumpL instructions use Ds[n], only the horizontal movement segment is considered. When JumpL instruction uses S[n], only the horizontal movement segment is considered.
- When Out(No,value,T[n]) is used, the timer will be reset after pause.
- When two motion instructions with the same points are executed consecutively in the program, the second motion instruction is invalid.
- When CP motion instructions (Movl/Movc/JumpL/Movs) are executed near the axis limit of a joint, false alarms of triggering axis limit in the planned trajectory may occur. This is a specification related issue, and the recommended actions include avoiding the axis limit or using PTP motion instructions (Movj/MovAbsJ/Jump) instead.

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

MH: Height of the highest point relative to the zero point of the base frame in the running process, in a value range of -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) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter**:

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.
7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.

8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.
9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:
    - The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.
    - When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.

11. For a Jump instruction with SLOn/SLReset, by default the robot automatically determines whether the JumpL instruction is suitable for segmentwise self-learning based on the motion parameters of the instruction:
    - The robot necessarily stops at the end of the third (descending) segment where it performs self-learning. In addition to this, the robot determines whether additional self-learning is needed at the end of the first (ascending) segment and second (horizontal) segment of the JumpL instruction.
    - If segmentwise self-learning is not needed, you can set the self-learning mode of Jump motion through the SetJumpSLMode instruction. For details, see the SetJumpSLMode instruction.

- **Example:**
    - 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
    - Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
    - Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

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

- The JumpL instruction applies to only the SCARA and Delta robots.
- Because the zero point of the base frame of the SCARA robot is above, MH is negative in most cases.
- Generally, the height parameter must satisfy MH > start point height + LH and MH > end point height + RH. For any settings that do not satisfy these conditions, a non-gate shaped motion may occur, which can be used for special applications.

Example:

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

4.7 Home

- **Function**: Returns to the origin of robot.
- **Description**: Returns to the work origin. You can specify the speed of motion during the returning process.
- **Format**: Home[Index],[V],[SLOn/SLOff/SLReset]
- **Parameter**:

Index: Work origin number, 0 to 4.

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

SLOn/SLOff/SLReset: (Optional parameter) Self-learning vibration suppression flag

- **Function**: It marks motions that require self-learning to suppress vibration.
- **Description**: This parameter is the default parameter of motion instruction, marking motions that require self-learning to suppress vibration.
- **Parameter:**

1. 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. In case of SLOn, self-learning is carried out only when the global motion speed is equal to or greater than 49%. In case of SLReset, self-learning is carried out regardless of the global motion speed.
2. The time for each self-learning is approximately 1.5s.
3. In case of SLOn, when the instruction is executed for the first time, the robot stops for about 1.5s after it reaches the target, automatically learning the relevant information and saving it in the learning data file. Normally, the information required can be learned automatically after one run. Therefore, the robot does not need to stop again to learn when the instruction is called repeatedly later. If the information required is not acquired by the first run, the self-learning will be performed again until the information required is acquired.
4. In case of SLReset, the robot stops for about 1.5s each time it reaches the target to automatically learn relevant information and save it in the learning data file.
5. If the vibration suppression effect cannot be met after self learning with the SLOn parameter, you can use the SLReset parameter, which is equivalent to forcing self-learning every time the motion is executed. Generally, it is only used for commissioning. When it is confirmed that the vibration meets the requirements, change SLReset to SLOn or SLOff.
6. For motion segments with good vibration conditions, do not mark them (SLOff). Otherwise, the robot may stop for self-learning when it first runs to the marked positions.
7. Generally, the more obvious the vibration, the better the likelihood of self-learning effect. However, if the vibration is too obvious, it may lead to self-learning of incorrect data.
8. For a motion instruction with SLOn or SLReset parameter, if self-learning is needed, the subsequent instructions will not be executed before the motion instruction is complete. In this case, the transition parameters do not take effect while the Nwait parameter takes effect. For a motion instruction with SLOn parameter, if self-learning is already completed, the transition parameters take effect. That is, the motion instruction with SLOn parameter supports transition after the self-learning is completed.
9. For a motion instruction with Until and SLOn/SLReset parameters at the same time, if Until is not triggered, the instruction moves normally to the target point where the robot stops and performs self-learning. If the Until is triggered during self-learning, the robot stops self-learning and executes the next motion instruction. If the Until is triggered during execution of the instruction, the robot does not perform self-learning.
10. In the following special conditions, the instruction that has completed self-learning may perform self-learning again:
    - The user clicks stop when the instruction is running, and the robot stops near the target point. Then the user clicks start and the robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - The user disables the robot servo when the instruction is running. The robot stops near the target point. Then the user enables the robot servo and click start. The robot executes the next instruction directly. If the next instruction includes SLOn, the next instruction will perform self-learning again.
    - Whether the previous instruction triggers Until or not affects the start point of the subsequent instruction containing SLOn. Therefore, if the subsequent instruction has completed self-learning when the previous instruction has not triggered Until, the subsequent instruction performs self-learning again when Until is triggered.

- When 0x1040, 0x1041, and 0x1042 errors occur during self-learning, the learning data of the current running instruction may not be saved successfully. After the robot is powered off and restarted, the motion instruction may need to perform self-learning again. It is recommended to perform self-learning again after the robot is powered off and restarted.

- **Example:**
  - 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 previously, self-learning will be performed when moving to LP[1] and LP[2]. The motion to the LP[2] continues self-learning before jumping out of the for loop.
  - Since the self-learning parameter is SLOff or default, self-learning will not occur when moving to LP[0] and LP[3].
  - Due to the presence of the SLReset parameter, the motion to LP[2] will not have transition effects. When the self-learning is completed at LP[1], the transition parameter for motion to LP[1] will produce a transition effects.

```
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 the work origin#2 (i.e., the third work origin) at 30% of maximum speed.
```

Output signal triggered by the working home: If the parameters triggering this function have been configured before the program runs the home point instruction, a signal will be output when the robot approaching the specified work home.

- **Example:**

```
Home[0],V[30];          ##Returns to the work origin#0 (i.e., the first work origin) at 30% of maximum speed.
```

If the output signal setting is configured for the work origin#0, when this instruction is executed in the program, the Out[1] signal is set to On when the work home#0 is reached.

- **Meaning of output trigger parameters:**

1. Work mode: There are three work modes, including:
   - Disable auto-recognition and don't signal: This function is disabled.
   - Enable automatic recognition J1-J6, and send a signal when approaching the origin: It only detects whether the manipulator axes fall within the error range of the origin. If yes, the corresponding Out signal is set to ON.

- Enable automatic recognition J1-J6, E1-E6, and send a signal when approaching the origin: When both the manipulator axes and external axes fall within the error range of the origin, the corresponding Out signal is set to ON.

2. Output port number: Specifies the number of the Out signal. The range is: NULL or Out[0]-Out[127] or Out[512]-Out[4607].
3. Offset range of axes and external axes: Set the offset range for each axis. The range is ±[0.01-10].

CAUTION

- If the Home instruction is used in the project file and the parameters related to the Home instruction are modified after running is suspended, the modification will not trigger a recompilation.
- If the modified line is before the running line or after the start line, the start line number and running line number are not affected. If the modified line is between the running line and the start line, the start line number and running line number are not affected, but the robot still moves according to the original planned trajectory during the actual execution of the program. This means that the modified Home instruction-related parameters will not take effect until the next execution of the instruction.
- The Home instruction does not take effect on independent axes.

4.8 ArmChange

- **Function**: Switches the arm type of the SCARA robot.
- **Description**: Switches between left and right arm types of the SCARA robot.
- **Format**: ArmChange value,V.
- **Parameter**:

Value: Arm pose, which can be -1, 1, or 0.

  - -1 indicates the left arm.
  - 1 indicates the right arm.
  - 0 indicates automatic switchover to the opposite of the current arm type.

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

CAUTION

- The transition of ArmChange is Z[0].
- When the SCARA robot is near a singularity (the angle of the J2 axis is less than 0.5°), it cannot switch arm type. Otherwise, an error occurs.

- **Example:**

ArmChange 1,V[20];

When the motion instruction before the ArmChange instruction does not include Nwait, the ArmChange instruction is not pre-compiled. This means that in this case, transition is not applicable to the ArmChange instruction. In the program shown in the following figure, it will only be pre-compiled to line 5.

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

4.9 Movs

- **Function**: Curve interpolation.
- **Description**: Uses curves to fit the spatial point sequence input by the user.
- **Format**: Movs P[N:M],V,Z,Tool,[Wobj],[Acc],[Dec],[NWait],[CurveParm], [Out(No,Val,P)]; **Parameter:**

P[N:M]: Required, N:M indicates the sequence of consecutive target points that can be reached from point N to point M in an ascending order. 4 ≤ total number of points ≤ 500. Common point form: P[1], LP[1], etc. One Movs motion instruction can contain only one point form, and does not support offset form and pallet getting point form.

V/Speed: Specifies a speed, required. For example, V[50] represents 50% of the maximum speed. Speed indicates the end linear speed. For details, see the 6.17 Speed instruction.

Z: Arrival and transition parameter, required. 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.

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 x transition unit length. (To set the transition unit length, go to "Settings" > "More" > "Motion parameters" > "Run parameter settings" > "Transition feature").

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/MovAbsJ/Movl/Movc, when the relative transition level m is equal to 100, it indicates that the maximum allowable transition length is half the motion length of the instruction, 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 the motion length of the instruction. When the ZR parameter is 200, the maximum allowable transition length is equal to the full motion length of the instruction.

2. If the current instruction is Jump/JumpL and the additional parameter RH=0, the maximum allowable transition length will exceed half the motion length 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 equal to the full motion length of the instruction.
3. If the current instruction is Jump/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].

- 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.
- Since the ZR parameter allows for a transition length longer than half the motion 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 motion length 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 the motion length of the instruction, they will be reduced to half the motion length of the instruction at the same time.
- 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 the motion length of the instruction), 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 instructions is large. The robot may transition in an asymmetric transition path within the transition area specified by the user.
- Note: The maximum distance for transition of the free curve does not exceed the first four points or the last four points.

For details, see Transition characteristics.

Tool: Tool frame, range Tool[0] to Tool[15], required.

For details, see Extended use of Tool parameters.

Wobj: Workobject frame, range Wobj[0] to Wobj[15], optional parameter.

For details, see Extended use of Wobj parameters.

When not specified, the default Wobj[0] is used.

In the tracking process, this parameter shall be replaced with Cnv. It is a mandatory parameter and is used to indicate motion in the tracking process.

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

When not specified, Acc is the same with V value by default. The minimum effective value is 20%.

When not specified and Speed is used, Acc is 20% by default.

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

When not specified, Dec is the same with Acc value by default. The minimum effective value is 20%.

NWait: (Optional parameter) No wait flag.

Indicates to execute the subsequent non-motion instructions immediately before the target point is reached.

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

- If the subsequent instruction is a motion instruction, regardless of whether this motion instruction includes NWait, the motion instruction will be issued in advance.
- In the case where multiple motion instructions are used, when the last motion instruction includes NWait, the non-motion instructions after the last motion instruction will be executed in advance when the first motion instruction is executed, regardless of whether the first motion instruction includes NWait.
- 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.

CurveParm: (Optional parameter) User-defined CurveParm type parameters and curve property mediation parameters. For details, see the user description of curve function.

```
{
    int PosPress;              #Position stress coefficient, default value 10, range 1 to 100.
    int OrientPress;           #Orientation stress coefficient, default value 10, range 1 to 100.
    int CurveMode;             #Curve mode, 0 default mode, 1 smoothing mode, 2 precision mode.
    double PosErr;             #Basic position error, default value 1 mm, range 0.1 mm to 100 mm.
    double OrientErr;          #Basic orientation error, default value 1°, range 0.1° to 10°.
    double CorneThre:    # Corner determination threshold, default value 90°, range 45° to 135°.
    double SmoErrRate;   #Corner smoothing error rate, default value 10, range 1 to 100.
    int TrajRecord;            #Trajectory record switch, default value 0 (OFF), range 0 to 1.
    int FilterLevel;           #Filter level, default value 0, range 0 to 3.
}
```

Out (No, value, P)…: (Optional parameter) Parallel output signal during operation. A maximum of three signals can be output in parallel in one Movs motion instruction. If the number of out signals exceeds three or if the bound points do not belong to the corresponding Movs instruction, an error occurs.

- **Subparameter:**

No: Output signal number, only supports 0-127.

Value: Output signal value.

P: The point number that triggers the signal output.

CAUTION

- Precision cannot be guaranteed for motion I/Os set in the transition area.
- The I/O triggering accuracy is different in different curve modes. For details, see the description of the free curve.
- When CP motion instructions (Movl/Movc/JumpL/Movs) are executed near the axis limit of a joint, false alarms of triggering axis limit in the planned trajectory may occur. This is a specification related issue, and the recommended actions include avoiding the axis limit or using PTP motion instructions (Movj/MovAbsJ/Jump) instead.

5 Signal Processing Instructions

5.1 Set

- **Format 1**: Set Out.
- **Function**: Output.
- **Description**: Sets a single digital output signal.
- **Format**: Set Out,value.
- **Parameter**:

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

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

- **Example:**

```
Set  Out[1],ON;          #Sets Out[1] to ON.
```

- **Format 2**: Set OG.
- **Function**: OG output.
- **Description**: Sets a group of digital output signals.
- **Format**: Set OG, BValue.
- **Parameter**:

OG: Output signal group, OG[0] to OG[15]. Elements defined by OG[Index] are subject to the Group instruction.

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

NOTE

- Before using OG, you must use the Group instruction to define the I/O elements corresponding to the OG. Otherwise, an error occurs indicating that OG is not defined it will report an error that the OG is not defined.
- The variable value is an integer in the range of [0,255], which is exactly an 8-bit binary number, each bit from right to left representing 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.

- **Parameter**:

DA: Analog output signal, DA[0] to DA[15], DA[64] to DA[79].

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

Determines whether the signal is current or voltage based on the configuration of the IR-Link or EtherCAT module. The input is limited according to 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.
```

When editing the instruction, if the value in the instruction exceeds the actual configured DA value range of 0 to 15 for IR-Link module or 64 to 79 for EtherCAT module, a prompt is given. If I/O instructions exist in the program and the configuration of the IR-Link module or EtherCAT module is subsequently modified, only the limit values of the actual configuration of the IR-Link or EtherCAT at startup will be taken. For details of the IR-Link module or EtherCAT module, see InoRobotLab Industrial Robot User Guide.

5.2 Get

- **Format 1**: Get In.
- **Function**: Reads In.
- **Description**: Reads a single digital input signal.
- **Format**: Get In[***],Var.
- **Parameter**:

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

Var: A variable that 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 Out.
- **Description**: Reads a single digital output signal.
- **Format**: Get Out[***],Var.
- **Parameter**:

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

Var: A variable that saves the read result. 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 IG.
- **Description**: Reads a group of digital input signals.
- **Format**: Get IG[***],Var.
- **Parameter**:

IG: Input signal group, IG[0] to IG[15]. Elements defined by IG[Index] are subject to the Group instruction.

Var: A variable that saves the read results. It represents a set of signal states.

NOTE

- Before using IG, you must use the Group instruction to define the I/O elements corresponding to the IG. Otherwise, an error occurs indicating that IG is not defined it will report an error that the OG is not defined.
- The value stored in Var is an integer in the range of [0,255], which is exactly an 8-bit binary number, each bit from right to left representing a signal state from low to high.

- **Example:**

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

Get IG[1],B[1];

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

##If signals In[7] to In[0] are OFF, OFF, OFF, OFF, OFF, ON, ON, ON successively, then B[1]=7.

- **Format 4**: Get OG.
- **Function**: Reads OG.
- **Description**: Reads a group of digital output signals.
- **Format**: Get OG[***],Var.
- **Parameter**:

OG: Output signal group, OG[0] to OG[15]. Elements defined by OG[Index] are subject to the Group instruction.

Var: A variable that saves the read results. It represents a set of signal states.

NOTE

- Before using OG, you must use the Group instruction to define the I/O elements corresponding to the OG. Otherwise, an error occurs indicating that OG is not defined it will report an error that the OG is not defined.
- The value stored in Var is an integer between 0 and 255, which is exactly an 8-bit binary number, each bit from right to left representing a signal state from low to high.

- **Example:**

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

Get OG [1],B[1];

##Reads the signal of OG[1];

##If signals Out[7] to Out[0] are OFF, OFF, OFF, OFF, OFF, ON, ON, ON successively, then B[1]=7.

- **Format 5**: Get AD.
- **Function**: Reads AD.
- **Description**: Reads analog input signal.
- **Format**: Get AD[***],Var.

- **Parameter**:

AD: Analog input signal, AD[0] to AD[15], AD[64] to AD[79].

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

Automatically determines whether the signal is current or voltage based on the configuration of the IR-Link or EtherCAT module.

Example:

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

5.3 Wait

- **Function**: The Wait instruction is used to wait for an event or time to be established. Otherwise, the task remains in the waiting state.
- **Description**: Waits until the input and output signals are detected to meet the conditions.
- **Format**:

Wait Conditional inequality.

Description: The next line of instruction is executed when the wait conditional inequality holds.

Wait T[Time];

Description: The next line of instruction is executed after waiting for a specified period of time.

Wait Conditional inequality, T[Time];

Description: The next line of instruction is executed when the wait conditional inequality does not hold within the specified time. An error occurs upon timeout.

Time: Time value, double type data, range 0 to 65535. Unit: s. Time=0 means that whether the inequality holds is determined for one time only.

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

Description: The next line of instruction is executed when the wait conditional inequality does not hold within the specified time. Otherwise, the program jumps to the line where the label L[i] is located.

Time: Time value, double type data, range 0 to 65535. Unit: s. Time=0 means that whether the inequality holds is determined for one time only.

Parameters (Variables are supported):

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

Numeric expression Inequality sign Numeric expression

T[Time] (Optional): Range 0.00 to 65535.00s.

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

- **Example 1:**

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

NOTE

Waits until the signal of input port [6] is ON before executing the subsequent instructions. The wait time is 10s at most. If the condition is still not met after 10s, an error occurs.

- **Example 2:**

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

##Executes the subsequent program lines

L[1]:

##Error handling

NOTE

Waits until the value of the B[2] variable is greater than 5 before executing the subsequent instructions. The wait time is 2s at most. If the condition is still not met after 2s, the program jumps to L[1].

- **Example 3:**

The time to wait again after the pause is equal to 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 lines

L[1]:

##Error handling

NOTE

Waits until the value of the B[2] variable is greater than 5 before executing the subsequent instructions. The wait time is 2s at most. The program is paused after waiting for 1s and the resumed. If the condition is still not met after 1s, the program jumps to L[1].

5.4 Group

- **Function**: Configures signal group.
- **Description**: Configures input and output signal group. A signal group contains up to 8 signals. Group I/O outputs can be defined using the Group instruction. The number of signals in a group can be smaller than 8. Note that IG and OG must be defined using the Group instruction before they can be used. Once the Group instruction is executed, the elements in the IG and OG will be cleared only when the controller is powered on again. In other cases, as long as the Group instruction is executed, the corresponding IG or OG will be recombined. If the Group instruction 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;

- **Parameter:**

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 the Group instruction takes effect once it is executed. If the Group instruction has been executed in a previous project, it can be used directly in a new project without causing any errors.

Restoration of default: Restored to undefined state upon re-power.

- **Example:**

Group OG[0] 1,2,7; ##Redefines OG[0] as Out[1], Out[2] and Out[7]. The I/O number can only be 0 to 127.

5.5 Invert

- **Function**: Inverts signal.
- **Description**: Inverts output signal.
- **Format**: Invert Out.
- **Parameter**:

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

- **Example:**

Set Out[1],ON;

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

The Invert instruction has a delay. If you need to read the signal value after the signal is inverted, wait for 1s before reading.

5.6 Pulse

- **Function**: Pulse instruction.
- **Description**: Triggers a pulse of a predefined width.
- **Format**: Pulse Out[*],T[*],High.
- **Parameter**:

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

  T: Pulse time width, default value: 0.2s, T[0.1] to T[2000].

  High: Outputs high pulses only. By default, pulses opposite to the current pulse are output.

T[*] and High are optional parameters.

- **Example:**

1. Pulse Out[1],T[10],High; #Sets Out[1] to output ON, and OFF after 10s
2. Pulse Out[1],T[10]; #Sets Out[1] to output state opposite to the current state, and the original state after 10s.
3. Pulse Out[1], T[10];
   Movj p[0],V[30],Z[0],Tool[0],Wobj[0];

   Pulse Out[1], T[10];

   The two pulses triggering the opposite levels overlap. The first pulse is interrupted by the second pulse.
4. Pulse Out[0], T[10];
   Wait T[2];

   Set Out[0],OFF;

   Suppose that currently Out[0] is OFF. After the pulse instruction is executed, Out[0] becomes ON. If the Set Out[0],OFF instruction is executed before the pulse instruction is completed, Out[0] will become OFF in advance. However, since the pulse instruction is not completed, it is necessary to wait for the specified time before Out[0] is set to OFF.
5. Pulse Out[50],T[10],High;

   Movj p[0],V[30],Z[0],Tool[0],Wobj[0],Out(50,OFF,DS[0]);

   Suppose that currently Out[50] is OFF. After the pulse instruction is executed, Out[50] becomes ON. If a motion instruction is executed before the pulse instruction is completed, Out[50] will become OFF in advance. However, since the pulse instruction is not completed, it is necessary to wait for the specified time before Out[50] is set to OFF.

- The pulse instructions are executed in parallel. After the output signal changes, the subsequent waiting process and signal reset will not affect the execution of subsequent programs. However, the output of the I/O signal is still affected by the motion I/O, Set instruction, and forced output configured in the InoRobotLab software.
- The number of pulses that execute different I/Os in parallel is limited to 64. That is, the maximum number of I/Os that are simultaneously running is 64.
- Once the pulse starts, it is not affected by program stop or emergency stop. The pulse continues to be executed until the end even in case of stop or emergency stop.
- The pulse instruction supports multitasking.
- The pulse accuracy is 100 ms, and the largest pulse instruction triggering delay is 200 ms.
- Out[*] and T[*] support custom variable input and BRD variable input.
- It is recommended to include High in the pulse instruction. It is not recommended to use continuous pulse instructions (no interval) for the same I/O.

5.7 BindTcpSpeed

- **Function**: Binds the linear speed analog to the analog output signal DA.

- **Description**: This instruction is used to bind the linear speed analog to the analog output signal.
- **Format**: BindTcpSpeed DA[0], minValue, maxValue, minSpeed,maxSpeed.
- **Parameter:**

DA: Analog output signal; Parameter range: Determined by the number of active DA modules (0 to (num*4-1)); Default: DA[0]; Variable input and label input supported.

minValue: The minimum value of the analog output; Data type: Double; Unit: Voltage V/mA; Parameter range: DA setpoint (0V to 5V, 0V to 10V, -5V to 5V, -10V to 10V, 0mA to 20mA), smaller than maxValue; Default value: 0; Variable input and label input supported.

maxValue: The maximum value of the analog output; Data type: Double; Unit: Voltage V/mA; Parameter range: DA setpoint (0V to 5V, 0V to 10V, -5V to 5V, -10V to 10V, 0mA to 20mA); Default: 0; Variable input and label input supported.

minSpeed: The minimum speed corresponding to the minimum analog value; Data type: Double; Unit: mm /s; Parameter range: 0 mm/s to maxSpeed; Default value: 0; Variable input and label input supported.

maxSpeed: The maximum speed corresponding to the maximum analog value; Data type: Double; Unit: mm/s; Parameter range: 0 mm/s to 100,000 mm/s; Default value: 0; Variable input and label input supported.

NOTE

After entering an interrupt, the linear speed analog and the analog output signal will be unbound.

- **Example:**

```
BindTcpSpeed DA[0],0,5,0,500;  #DA outputs data during motion
Movj P[1],V[30],Z[0],Tool[1];
Movj P[2],V[30],Z[0],Tool[1];
CloseBindTcpSpeed DA[0];
```

5.8 CloseBindTcpSpeed

- **Function**: Unbinds the linear speed analog from the analog output signal.
- **Description**: This instruction is used to unbind the linear speed analog from the analog output signal.
- **Format**: CloseBindTcpSpeed DA[0].
- **Parameter**:

DA[***]: Analog output signal; Parameter range: Determined by the number of active DA modules (0 to (num*4-1)); Default: DA[0]; Variable input and label input supported.

- **Example**:

```
BindTcpSpeed DA[0],0,5,0,500;  #DA outputs data during motion.
Movj P[1],V[30],Z[0],Tool[1];
Movj P[2],V[30],Z[0],Tool[1];
CloseBindTcpSpeed DA[0];
```

6 System Parameters-Related Instructions

6.1 SetAccRamp

- **Description**: Sets the acceleration and deceleration jerk levels of the motion segment.
- **Format**: SetAccRamp(StartValue,EndValue).
- **Parameter**:

StartValue: Acceleration jerk percentage of the start segment. Integer data, range 1 to 100.

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

- This instruction is effective for Movj, Movl, Movc, Jump, JumpL, ArmChange, Home and MovAbsJ motions and will affect the acceleration jerk in these motions.
- When the instruction SetAccRamp is not used, the system default acceleration jerk parameter is dependent on the robot model.
- The scope of SetAccRamp is the whole system. For initialization conditions, see Scope of Variables and Instructions

6.2 SlewMode

- **Function**: Sets 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.
- **Parameter**:

nMode: Rotation optimization mode, 0, 1, 2, 3

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

  - 0: No optimization, subject to arm parameters in the position variable.
  - 1: Optimization mode 1 to ensure that J4 is always within the range of (-180,180] during motion.
  - 2: Optimization mode 2 to ensure that J4 always moves in the shortest way during motion. That is, determine whether reaching the target point requires J4 to rotate by 180 degrees. If the angle difference is ≤180°, the target point is fully reached. If >180°, 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 motion 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. 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; ##Disables optimization

……

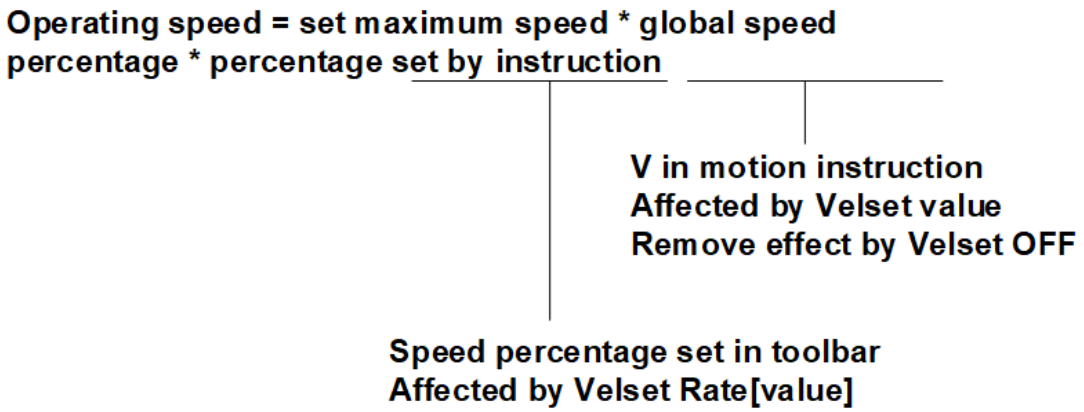
CAUTION

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

For initialization conditions, see 2.4 Scope of Variables and Instructions.

6.3 Velset

- **Function**: Sets speed.
- **Description**: There are two ways to set the global speed: "Velset Rate[value]" to set the global speed percentage and "Velset value" to force the speed instruction value. Once set, "Velset value" remains in effect until it is reset or "Velset OFF" is encountered.



- **Format**:

Format 1: Velset Rate[value];

Parameter:

Rate[value]: Global speed percentage.

NOTE

Sets the global speed percentage. When you run this instruction, the global speed percentage changes.

Format 2: Velset value;

Parameter:

Value: Forces the speed instruction value as a percentage.

NOTE

The V parameter after the Move and Jump series instructions is ignored, and Value is always used as the percentage set by the instruction. (Note: Velset value instruction has no effect on the Speed instruction.)

VelSet value is a macro-defined processing mode, which is not affected by program logic. That is, as long as the instruction exists, even if it is in an If-else statement that is not executed, it will still affect the subsequent instructions. The "subsequent instructions" here means the program lines after the If-else statement.

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

Format 3: Velset OFF;

Cancels the speed setting of "Velset value". The speed setting in motion instructions comes into effect.

- **Example:**

##Assume that the current speed setting is 100%.

START;

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

Velset Rate [50];

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

Velset 30;

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

Velset OFF;

END;

6.4 GetAlarmNo

- **Description**: Gets alarm number and saves value in specified variable.
- **Format**: GetAlarmNo(Var).
- **Parameter**:

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]);

CAUTION

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.

6.5 GetRunState

- **Description**: Gets the system operating status and stores the value in the specified variable.
- **Format**: GetRunState(Var).
- **Parameter**:
  - Var: Numeric variable (integer type) used to store system operation status.
  - 0: Stop.
  - 1: Start.
  - 2: Step into.
- **Example**:

```
GetRunState(B[1]);
Switch B[1]
    Case 0:
Print "State is stop";
            Break;
    Case 1:
Print "State is run";
            Break;
    Case 2:
Print "State is step";
            Break;
    Default:
Print "State is undefined"
Break;
EndSwitch;
```

6.6 SetSysToolNo

- **Description**: Sets the tool number used by the system.
- **Format**: SetSysToolNo(ToolNo).
- **Parameter**: ToolNo: Tool number, integer, range 0 to 15.
- **Example**:

```
SetSysToolNo(4);
##The tool number used by the system in the upper right corner of the teach pendant has changed to "4".
```

If the tool number to be switched is not Tool 0 and the load mass is 0, the switchover fails.

6.7 SetSysWobjNo

- **Description**: Sets the workobject number used by the system.
- **Format**: SetSysWobjNo(WobjNo).
- **Parameter**:

WobjNo: Workobject number, integer value, range 0 to 15.

- **Example**:

SetSysWobjNo(4);

##The workobject number used by the system in the upper right corner of the teach pendant has changed to "4".

6.8 Clear

- **Description**: Clears the value of the object. The object can be a numeric variable, an offset variable, a tool frame, a workobject frame, a hand-held workobject, 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.
- **Parameter**:

Object: It can be any of the following types of objects.

Target	Description
Tool	Clears the tool frame parameters, including tool load parameters, Tool[0] to Tool[15]. Only TLoad/TFrame parameters are restored. Other parameters remain unchanged.
Wobj	Clears the workobject frame parameters, Wobj[0] to Wobj[15]. Only UFrame/OFrame parameters are restored. Other parameters remain unchanged.
Load	Clears the hand-held load parameters, Load[0] to Load[15].
Out	Clears the Out variable, Out[0] to Out[13823].
Fin	Clears the state of the In variable to "not forced". Only Fin[0] to Fin[63] are supported.
Box	Clears the interference zone setting Box[0] to Box[15] in the instruction.
B/LB	Clears the B/LB variable.
R/LR	Clears the R/LR variable.
D/LD	Clears the D/LD variable.

Target	Description
PR/LPR	Clears the PR/LPR variable.
Alarm	Clears the alarms (For use in static task)
Port	Clears the port data.

Num: Number of objects to be cleared Clears multiple variables consecutively starting from Object.

- **Format 2**: Clear All

Clears all variables, including all the above-mentioned custom interference zones BOX and the variables LB /LR/LD/LPR in the current module. Alarms and all open port data are also cleared.

Clearing Tool/Wobj/Load is to restore the variable values set in the system parameters. It does not set the object values to zero. In the program, you can change the tool, user, and work object parameters using instructions, and restore the original values through the instruction Clear.

Clearing Out is to set the Out signal to OFF.

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

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

Clearing Port is to clear the port data.

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

Clearing alarms is used in static tasks.

When an alarm occurs during running, 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.

- **Example**:

Clear Tool[1],2;

Clear Out[4],3;

Clear PR[2],4;

6.9 SetFlyWait

- **Function**: Turns on or off the transition function for non-motion instructions.
- **Format**: SetFlyWait (ON/OFF).
- **Parameter**: OFF: Turns off transition wait mode (default), ON: Turns on transition wait mode.

CAUTION

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 motion 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≠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];

Analysis:

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].

Example 2:

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

Set Out[4],On

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

Analysis:

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

Since the transition length has reached half the motion length of the instruction, 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];

Analysis:

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].

CAUTION

- This function is not supported in dynamic frames.
- When the subsequent instruction is one of the following instructions, the robot waits for motion to stop before executing the instruction.
  - Pause
  - END
- In general, when a given transition length is greater than half the motion length of the current instruction, the robot starts executing subsequent non-motion instructions when half the motion length of the current instruction is reached. 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 early as it reaches half the motion length of the current instruction.
- If the current instruction is Jump/JumpL with a vertical descending segment (RH≠0), the robot will execute subsequent non-motion instructions no earlier than the vertical descending begins.
- 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).
- 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).
- 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.
  - 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.
  - 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 effective in the main task.
2. This function is not supported in dynamic frames. The robot executes subsequent non-motion instructions after the motion has stopped.

3. This function is not supported in the fixed-path transition mode. When NWait is not enabled in the fixed-path transition 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 2.4 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 point P[0], 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.

After the robot enters the transition area (Z[CP] is half the instruction length), the robot keeps moving and starts to execute subsequent non-motion instruction Set Out[5]On simultaneously.

6.10 SetAccuracyMode

- **Function**: Sets the precision mode of the robot.
- **Format**: SetAccuracyMode(Normal/MidAccuracy/HighAccuracy).

SetAccuracyMode(Var); #Var is a variable.

- **Parameter**:

0 - Normal (default mode):

When the default mode is used, 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 medium accuracy mode is used, 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 currently not supported.

Accuracy settings only apply to free transition mode (default transition mode).

The differences between the modes are shown in the table below:

Mode	Category	Description
0 - Normal (Default mode)	Accuracy	Lowest, accuracy cannot be guaranteed during continuous motion.
	Efficiency	Highest
	Disadvantage	Accuracy cannot be guaranteed during continuous motion.
	Application	Applications with low accuracy requirements such as sorting and handling
1 - MidAccuracy (medium accuracy mode)	Accuracy	Higher than default mode
	Efficiency	Low efficiency
	Disadvantage	Low efficiency
	Application	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
	Disadvantage	Position stabilization time may be extended.
	Application	Applications with high requirements for trajectory

Mode	Category	Description
		accuracy such as high-speed printing
3 - LowSpeedHighAccuracy (low speed high accuracy mode)	Accuracy	High accuracy at low speeds
	Efficiency	Same with medium accuracy mode
	Disadvantage	This mode cannot be applied at high speeds as it easily causes vibration.
	Application	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

6.11 GetTorque

Function: Reads robot torque data.

Format: Double GetTorque(int AxisNo).

Parameter:

AxisNo: Axis number, range 1 to 6.

Scope of instruction: Only effective in the main task.

Return value: Robot torque data, double-type.

- **Example**:

#Read the current data value for the first axis.

D[6] = GetTorque(1);

6.12 RapidMove

- **Function**: Turns on or off the optimal trajectory function.
- **Format**: RapidMove(MotionType, Enable).
- **Parameter**:

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

When MotionType=CP, Enable=ON, optimal trajectory planning is enabled for the CP instructions such as Movl/Movc/JumpL.

When MotionType=CP, Enable=OFF, optimal trajectory planning is disabled for the CP instructions such as Movl/Movc/JumpL.

When MotionType=PTP, Enable=ON, optimal trajectory planning is enabled for the PTP instructions such as Movj/Jump/MovAbsJ.

When MotionType=PTP, Enable=OFF, optimal trajectory planning is disabled for the PTP instructions such as Movj/Jump/MovAbsJ.

When MotionType=ALL, Enable=ON, optimal trajectory planning is enabled for all motion instructions.

When MotionType=ALL, Enable=OFF, optimal trajectory planning is disabled for all motion instructions.

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. The optimal trajectory function is enabled for motion instructions between RapidMove(*, ON) and RapidMove(*, OFF).

- **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];
```

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, resulting in slow motion and errors.

6.13 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 specified in this instruction. Once set, this instruction remains in effect until SetAcc is executed again or the project is initialized.
- **Format**:

```
SetAcc(accValue,decValue);
```

SetAcc(OFF);

- **Parameter**:

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 parameters in Movj, Jump and other instructions are 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 Acc parameters in Movj, Jump and other instructions are replaced, with the SetAcc value as the acceleration percentage.

OFF: Cancels the acceleration setting of SetAcc(accValue, decValue). 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];

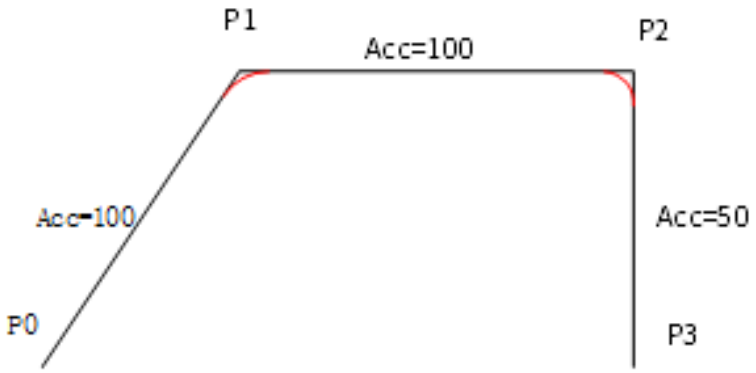
SetAcc (100,100);

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%.



6.14 SLVSMode

- **Function**: Sets the self-learning vibration control mode.
- **Description**: This instruction is used to turn self-learning vibration control function on or off.
- **Format**: SLVSMode modePara.
- **Parameter**:

  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 control function is turned on, and Off indicates that the vibration control function is turned off.
2. The higher the level of vibration control, the better the effect of vibration suppression. In terms of vibration amplitude, HighLevel<=MidLevel<=LowLevel, and in terms of motion execution time, HighLevel>=MidLevel>=LowLevel.
3. When the self-learning vibration control 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**:

In the following program, after the instruction SLVSMode MidLevel is executed, the self-learning vibration control 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],Tool[1],Wobj[0],SLOn;
    Delay T[1];
    Movj LP[1],V[30],Z[0],Tool[1],Wobj[0],SLOn;
    Delay T[1];
    SLVSMode MidLevel;
EndFor;
```

After the self-learning vibration control function is turned on, robot torque may exceed the limit during operation. Adjust the operation parameters appropriately; for example: before the self-learning vibration control function is turned on, the robot can execute motion instructions for the optimal trajectory normally. After the function is turned on, an alarm of overlimit torque may occur during operation.

6.15 SLDataClear

- **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.
- **Parameter**:

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 for the current position of the robot 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**:

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 statement 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],Tool[1],Wobj[0];
    Delay T[1];
    Movj LP[1],V[30],Z[0],Tool[1],Wobj[0],SLOn;
    Delay T[1];
    SLVSMode MidLevel;
EndFor;
```

6.16 SetJumpSLMode

- **Function**: Sets the self-learning mode for the Jump/JumpL instruction.
- **Description**: Sets whether to perform segmentwise self-learning on the Jump/JumpL instruction.
- **Format**: SetJumpSLMode(iMode).
- **Parameter**:

iMode:

Includes SL_Auto and SL_LastSeg.

SL_Auto: Automatic mode, which is the default mode. The robot automatically determines whether the Jump instruction is suitable for segmentwise self-learning based on the working conditions. The robot determines whether the first (ascending) segment and second (horizontal) segment are suitable for self-learning. If suitable, the robot stops at the end of the first and second segments respectively and performs self-learning.

SL_LastSeg: The robot performs self-learning at the last segment of motion, i.e., it stops at the target specified in the Jump instruction where it performs self-learning. This mode is intended for special situations where segmentwise learning is not needed.

- **Example**:

……

SetJumpSLMode(SL_LastSegment); ##Sets to perform self-learning only at the target specified in the Jump instruction

Jump P[1],V[30],Z[3],Tool[1],Wobj[1],LH[60],MH[-130],RH[60],SLOn; ##Performs self-learning only at P[1]

JumpL P[2],V[30],Z[3],Tool[1],Wobj[1],LH[60],MH[-130],RH[60],SLOn; ##Performs self-learning only at P[2]

SetJumpSLMode(SL_Auto); ##Sets automatic mode. The algorithm automatically determines whether to perform segmentwise self-learning.

……

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

For initialization conditions, see 2.4 Scope of Variables and Instructions.

6.17 Speed

- **Function**: Sets the TCP speed, TCP orientation speed, external axis linear speed, and external axis rotational speed for the robot to perform CP motion, and sets whether these speeds are affected by the global speed percentage.
- **Description**:

1. It allows for numerical setting of the TCP speed and orientation speed to meet the market's precise control requirements for processes such as grinding and gluing.
2. It allows you to flexibly set whether the TCP speed is affected by the global percentage to meet certain process requirements.
3. It allows you to directly modify the speed in a certain program line for quick fine tuning.

- **Format 1**: Sets robot speed using numerical values.

    Movl P[0], Speed[1000].Ori:500,Z[0],Tool[1];

    Movl P[0], Speed[1000],Z[0],Tool[1];

Parameter range: [0,15000]

Supported Variables: The instruction supports R/LR, B/LB variables, and integer and Byte type variables, but does not support D/LD variables and Double type variables.

- **Format 2**: Sets robot speed using the Speed struct.

  Speed SpeedParm= {1000,500,5000,1000,1};

  Movl P[1],Speed[SpeedParm],Z[0],Tool[0],Wobj[0];

  Movl P[2],Speed[SpeedParm],Z[0],Tool[0],Wobj[0];

- **Parameter**: Sets the TCP speed in TCP parameter, and the TCP orientation speed in Ori parameter. If there is an external axis, you can use Leax to set the linear speed of the external axis, and Reax to set the rotational speed of the external axis (Bstatic = 1 indicates that the speed is constant and not affected by the global percentage, Bstatic = 0 indicates that the speed is affected by the global percentage.)

  Struct Speed

{

TCP, ##TCP speed

Ori, ##TCP orientation speed

Leax, ##Linear speed of the external axis

Reax, ##Rotational speed of the external axis

Bstatic; ##Whether the speed is affected by the global percentage

}

Scope of the instruction: It is a parameter in the motion instruction and takes effect in the current motion instruction.

- **Parameter range**:

TCP: [0.0001,15000]

Ori: [0.0001, 36000]

Leax: [0.0001, 100000]

Reax: [0.0001, 3600000]

Bstatic: [0, 1]

- **Detailed description**:

1. You need to manually enter TCP speed. By default, v_ori = 500°/s, v_leax = 5000 mm/s, v_reax = 1000°/s, and bStatic = 0.
2. You can use Speed[1000].Ori:500 to modify the Speed parameter.
3. When Bstatic = 1, the next instruction with Bstatic = 0 is affected by the global percentage only after the transition is completed.
4. The speed values in the Speed instruction may not be reached when subject to physical constraints.
5. The Speed instruction must be used with the instruction SetAcc. If the instruction SetAcc is not used, the default acceleration is 20%.

- **Example**:

Movl P[0], Speed[100],Z[0],Tool[1]; ##The robot end trajectory speed is 100 mm/s, and the orientation speed is 500°/s.

Movl P[0],V[60],Z[0],Tool[1]; ##The actual speed is 60%.

Velset Rate[50];

Movl P[0],V[60],Z[0],Tool[1]; ##The actual speed is 50% x 60% = 30%.

Movl P[0], Speed[100],Z[0],Tool[1]; ##The TCP speed is 100 x 50% mm/s, and the orientation speed is 500 x 50% °/s.

Movl P[0], Speed[100].Bstatic:1,Z[0],Tool[1]; ##The TCP speed is 100 mm/s, and the orientation speed is 500°/s.

Velset 30; ##Speed setting is not affected by Velset 30.

Movl P[0],V[60],Z[0],Tool[1]; ##The actual speed is 30% x 50% = 15%.

Movl P[0],Speed[100],Z[0],Tool[1]; ##The TCP speed is 100 x 50% mm/s (Not affected by Velset 30).

SetAcc(100,100);

Movl P[0],Speed[100],Z[0],Tool[1]; ##The acceleration coefficient is 100%.

SetAcc(OFF);

Movl P[0],Speed[100],Z[0],Tool[1],Acc[50]; ##The acceleration coefficient is 50%.

6.18 GetLocalSysTime

- **Function**: Gets the controller local time.
- **Description**: Gets the controller local time.
- **Format**:

1. GetLocalSysTime(RVar).

2. Int Num = GetLocalSysTime(RVar).

- **Parameter**:

RVar: Only R/LR variables are allowed. It is used to store the time of execution of the instruction in order of year, month, day, hour, minute, and second in an array of variables with the R/LR variable as the starting address.

Return value: The number of saved values.

6.19 SingAreaHandle

- **Function**: Singularity crossing.
- **Description**: The 6-axis robot passes through the singular area of the wrist using CP instruction.
- **Format**: SingAreaHandle(Wrist,ON/OFF).

- **Parameter**:

  Wrist: Wrist singularity processing, default value, non-modifiable.

ON/OFF: Turns on/off the wrist singularity crossing function.

CAUTION

- When the singularity crossing function is enabled, all CP instructions after this instruction run in the singularity crossing mode (only the position accuracy of TCP is guaranteed) until the function is disabled by a new instruction, or the default value is restored (the default value is "Wrist,OFF").
- The default value is restored automatically under the following conditions:
  - The program has a file or point modification, which requires the controller to compile the project.
  - The program running pointer is return to the start line.
  - The program running pointer is set to another function.
  - The controller is powered on again.
- The transition instruction does not take effect between the motion instruction enabled with this function and the motion instruction that is not enabled with this function.
- After this function is enabled, a small portion of the trajectory cannot reach the orientation/joint angle of the target position taught.
- If the change in orientation cannot meet the requirements after the singularity crossing function is enabled, you can try to disable the function and change ArmType[4] of the first and last points to the same value.

Example:

In the following program, the CP motion instructions after the instruction "SingAreaHandle(Wrist,ON);" run in singularity crossing mode.

```
Start;
  SingAreaHandle(Wrist,ON);
  Movj P[1],V[100],Fine,Tool[1],Wobj[0];
Movl P[2],Speed[1000],Z[0],Tool[1],Wobj[0];
Movc P[3],Speed[1000], Z[0],Tool[1],Wobj[0];
Movc P[4],Speed[1000], Z[0],Tool[1],Wobj[0];
End;
```

6.20 SetVarRecordMode

- **Description**: Sets the recording range for the black box's process of tracking changes in project variables.
- **Format**: SetVarRecordMode(Flag, Value).
- **Parameter**:

Flag: The flag bit to be set. Integer data, range 0 to 3.

- 0: All variables.
- 1: Custom variables (including custom local variables, module variables, global variables).

- 2: Module variables (LB/LR/LD/LPR/LP/Lpallet).
- 3: Global variables (B/R/D/PR/P/JP/LOAD/TOOL/WOBJ/STR/Pallet).

Value: The switch corresponding to the flag bit. Integer data/ON/OFF, range 0 to 1.

- 0: Turn off the record.
- 1: Turn on the record.

- This instruction is used to set whether to enable the black box to record changes in project variables and to adjust the recording range. For details on viewing, saving, and exporting recorded content, see the System Diagnostics Function User Guide.
- Enabling the variable change recording switch will affect the execution efficiency of instructions such as variable definition/assignment.
- The recording switch is disabled by default. Once enabled using the instruction, it will remain effective until it is disabled using the instruction or the system is rebooted.

6.21 GetRobotType

- **Description**: Get the current robot model type.
- **Format**: String = GetRobotType( ).
- **Parameter**: /.
- **Return Value**: Robot model name string.
- **Example**:

```
Str[1] = GetRobotType( );
Str[2] = "IRS111-3-40Z15TS3-S0_01740222";
R[1] = Strcmp(Str[1] , Str[2]);
If R[1] <> 0
    Print "Model mismatch";
EndIf;
```

7 Process Control Instructions

7.1 Comment

- **Function**: Inputs comment.
- **Description**: Inserts a comment into the program.
- **Format**: # Content.
- **Parameter**:

Content: Content of comment.

- **Example**:
  ##Program is made at 10;

7.2 L-Goto

- **Function**: Logic 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[lableNo]: #Must be unique.

……

Goto L[lableNo];

- **Parameter**:

lableNo: Label number, an integer from 0 to 9999

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. For example, 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 instruction in the loop to add delay.

In addition: When label instructions are used, L and GOTO L[] cannot be used across functions, 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;

##Moves to P[0] and then performs reciprocating motion between P[1] and P[0].

7.3 Delay

- **Function**: Delay.
- **Description**: Program delay, controlled by parameter T.
- **Format**: Delay T[Var].
- **Parameter**:

Var: Time; Unit: s; Value range: 0.000 to 65535.000. The time accuracy is 0.1s.

Var cannot be Out, In, OutB, InB, OutW, and InW variables.

Example:

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

The Delay instruction is considered motion instruction in the tracking process.

7.4 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".
- **Parameter**: Module name.
- **Example**:

Include "Model1.pro"; Declares the reference to Model1.pro module.

Start;

Model1.fun1();

End;

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

7.5 Func

- **Function**: Custom functions and their calls.
- **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. (Recursive calls are not supported for Func.)

- **Format**:

  Format of the main function

Start;

…function body…

End;

Custom function format:

Func function name (parameter list)

…function body…

EndFunc;

Format of custom function with function return value: (Compatible with above custom functions without function return value)

Func <ReturnKind> <FuncName>(<parameters>)

…function body…

Return variable name;

EndFunc;

Custom function with function return value:

- **Function**: Custom function with function return value and its call.
- **Description**: A custom function can be returned with a return value.
- **Supported function return value types**: Byte, Bool, Int, Float, Double, String.

- When the function return value type is Byte, a variable of type Byte, B, and LB, or a constant of type Byte can be returned.
- When the function return value type is Bool, a variable of type Bool, or a constant of type Bool (True, False, 0, 1 supported) can be returned.
- When the function return value type is Int, a variable of type Int, R, and LR, or a constant of type Int can be returned.
- When the function return value type is Float, a variable of type Float, or a constant of type Float can be returned.
- When the function return value type is Double, a variable of type Double, D, and LD, or a constant of type Double can be returned.
- When the function return value type is String, a variable of type String, Str, or a constant of type String can be returned.

- **Parameter list**:

The parameter list can be customized variables of bool, byte, int, float, double, and string types, or arrays. Structs are not supported. The parameter list includes:

1. IN parameter
   Example:

   Func funAdd(int aa,int bb)

   R[1] = aa + bb;

   EndFunc;

2. OUT parameter
   Add the sign "&" before the variable definition to indicate that the variable can be passed out.

   Example:

   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 list); ##It is required to include the corresponding program name at the beginning 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 this 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 another module

ModelB. funDec(30,10);

End;

Submodule ModelB.pro:

Func funDec(int aa,int bb)

R[1] = aa - bb;

EndFunc;

Call format of custom function with function return value:

(1) Calling a custom function in the main module

```
Func Int AddTeach1(Int aa, Int bb)
Int cc = aa + bb;
Return cc;
EndFunc;
```

Call mode 1:

```
Int dd = AddTeach1(3,5);
```

Call mode 2:

```
Int ee;
ee = AddTeach1(4,6);
```

(2) Calling, by the main module, a custom function in other modules

Define this function in the "test.pro" module:

```
Func Int AddTeach2(Int aa, Int bb)
Int cc = aa + bb;
Return cc;
EndFunc;
```

Called in the main module:

Call mode 1:

```
Int dd = test.AddTeach2(3,5);
```

Call mode 2:

```
Int ee;
ee = test.AddTeach2(4,6);
```

7.6 Ret

- **Function**: Returns from the function.
- **Description**: Returns to the main function from another function. The subsequent contents of the another function are not executed any more. Instead, the statements of the main function are executed.
- **Format**: Ret.

CAUTION

- It is often used together with function calls.
- In the case that the start line is set to be executed from the main function:
  - If the Ret instruction is included in the main function, the running state will be set to end when the program runs to the program line where Ret is located.
  - If the Ret instruction is included in a function, the program returns from the function to the main function when it runs to the program line where Ret is located, and instructions after the Ret instruction in the function will no longer be executed.
- If the start line is set to be executed from the function, the running state will be set to end when the program runs to the program line where Ret is located.

7.7 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
    Stmt1;
ElseIf  condition2
    Stmt2;
ElseIf  condition3
    Stmt3;
……
Else
    Stmtn;
EndIf;
```

- **Parameter**:

Condition: Indicates condition.

Stmt: Statement executed when the condition is met.

NOTE

- Judgment conditions can be the combination of multiple conditions.
- Stmt 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;

7.8 For-EndFor

- **Function**: Loop statement with number of executions.
- **Description**: First, execute an initialization assignment statement and then determine whether the condition is met. If the condition is met, the contents in the loop are executed. After the execution is finished, the variable in the initialization statement increments by one stepsize. Then the condition is determined again and if the condition is met, repeat the previous steps, and so on, until the condition does not hold.
- **Format**:

    For InitExp, Condition, Step[n]

        Stmt;

    EndFor;

- **Parameter**:

InitExp: Initialization assignment statement for B/R/LB/LR variables

Condition: Indicates condition.

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

Stmt: Statement executed when the condition is met.

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. For example, 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 instruction in the loop to add delay.

- **Example**:

For B[0]=0,B[0]<5,Step[2]

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

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

EndFor;

> NOTE
>
> The two Movj instructions loop three times.

7.9 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:

Stmtn;

Break;

EndSwitch;

- **Parameter**:

Var: Variable, which can be an integer value variable or a string variable.

Value: Value of the variable. Can be an integer, or a string.

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.

CAUTION

- In general, it is best to end each Case (including the 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;
```

7.10 While-EndWhile

- **Function**: Conditional loop.
- **Description**: Determines whether the condition is met. If the condition is met, execute the statement within the loop. Once a condition is not satisfied, the program jumps out of the loop.
- **Format:**

```
While Condition
    Stmt;
EndWhile;
```

- **Parameter**:

Condition: Indicates condition.

Stmt: Statement executed when the condition is met.

NOTE

- Judgment conditions can be the combination of multiple conditions.
- Stmt can be several lines of instructions.

CAUTION

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. For example, 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 instruction in the loop to add delay.

- **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;
```

NOTE

The above instructions loop three times and the whole program is carried out three times between P[1] and P[2].

7.11 Break

- **Function**: Ends the loop.
- **Description**: Completely ends a loop to execute 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. 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;
##Execute \\"Movj P[0], Movj P[1]\\" twice.
```

7.12 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;
##Execute \\"Movj P[0], Movj P[1]\\" four times.
```

7.13 WaitInPos

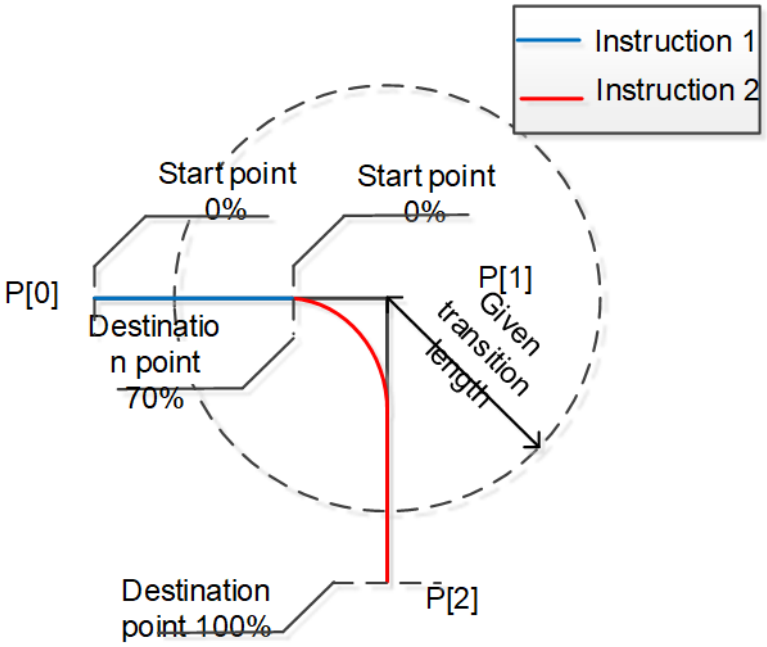
- **Description**: Waits until the motion reaches the specified proportion or the encoder position is reached.
- **Format**: WaitInPos(Value/Fine/Corner).
- **Parameter**:

1. Value: Percentage of current motion displacement (optional).
    - Parameter type: Integer.
    - Value range 1 to 100, default value 100.
    - Note: /.

2. Fine: Waits for current motion until encoder position is arrived (optional).
   Parameter type: Keyword.
3. Corner transition intermediate point (optional)
   - Parameter type: Keyword.
   - Note: Near the transition intermediate point of the previous motion and the current motion.

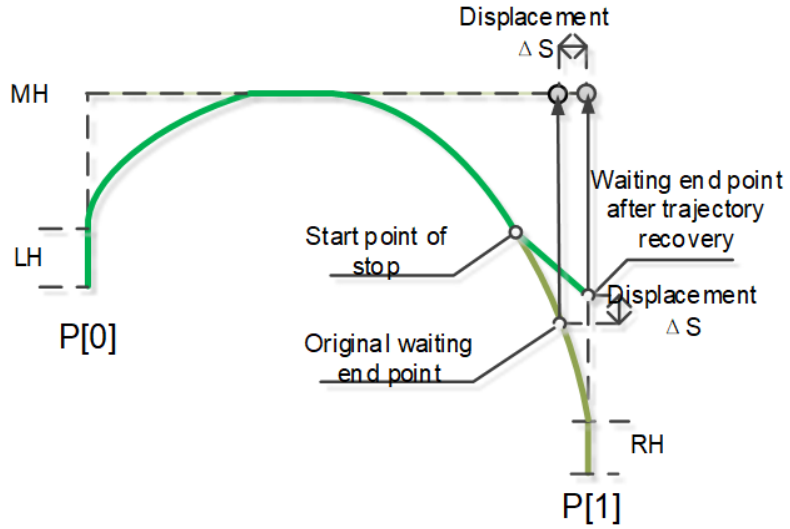
CAUTION

- This instruction can only be used in motion tasks.
- This instruction is only supported in static frames. In dynamic frames, the robot will wait for the motion to stop before executing subsequent instructions.
- 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.
- As the motion path of the transition area cannot be guaranteed, there may be deviation in automatic calculation of the transition intermediate point using the Corner parameters. Please adjust it according to the actual situation.
- The displacement percentage D is referenced to the previous motion. When the target points of multiple consecutive motion instructions are too close or duplicated, only the first motion instruction can produce motion. In this case, the robot takes the first motion instruction as reference and executes the subsequent instruction when the robot reaches the specified displacement percentage.
- When the current position of the robot is greater than or equal to the specified displacement percentage, the robot does not wait and directly executes the subsequent instructions. In particular, when a stop signal is received, the robot will no longer detect the current displacement percentage. Therefore, the subsequent instructions will not be triggered, until the robot resumes running from the stop state.
- The accuracy of WaitInPos(100) is Z[0]. Select different accuracy levels according to actual accuracy requirements.
- Note that when two consecutive motion instructions form a transition, the start 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 the figure below, the transition to instruction 2 begins when 70% of instruction 1 is reached. In this case, it is considered that instruction 1 ends at the transition start point from which instruction 2 starts running.



- When Corner is used 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 begins, and the robot immediately executes subsequent instructions.

- 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 also changes. If the robot reaches the displacement value corresponding to the original waiting end point when the Jump instruction is restarted after a stop, the robot may end the wait at a point diagonally above the original waiting end point.



- **Scope of instruction (including usage scenario limits, whether it is effective in multitasking)**

1. Only effective in the main task.
2. This function is not supported in the dynamic tracking.
3. The SetFlyWait setting is reset to the default value when one of the following conditions occurs.

It is subject to project-level variable reset conditions.

- **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];

Analysis:

When 30% of the motion is completed, determine In[5]==On and execute Set Out[4],On.

When 80% of the motion is completed, execute Set Out[5],On.

Transition is performed when passing point P[1]. The theoretical transition area is less than 20%.

Example 2:

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

WaitInPos(30);

B[0]=B[0]+1;

WaitInPos(80);

Set Out[5],On;

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

Analysis:

When 30% of the motion is completed, execute B[0] = B[0] + 1.

When 80% of the motion is completed, execute Set Out[5],On.

Transition is performed when passing point P[1]. The theoretical transition area is 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];

Analysis:

When 100% of the motion is completed, execute Set Out[4],On and Set Out[5],On.

When 100% of the motion is completed, and the encoder reaches the arrival accuracy range and stays for the set time, execute Set Out[6],On.

The robot stops briefly when the P[1] is reached, with no transition.

Example 4:

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

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

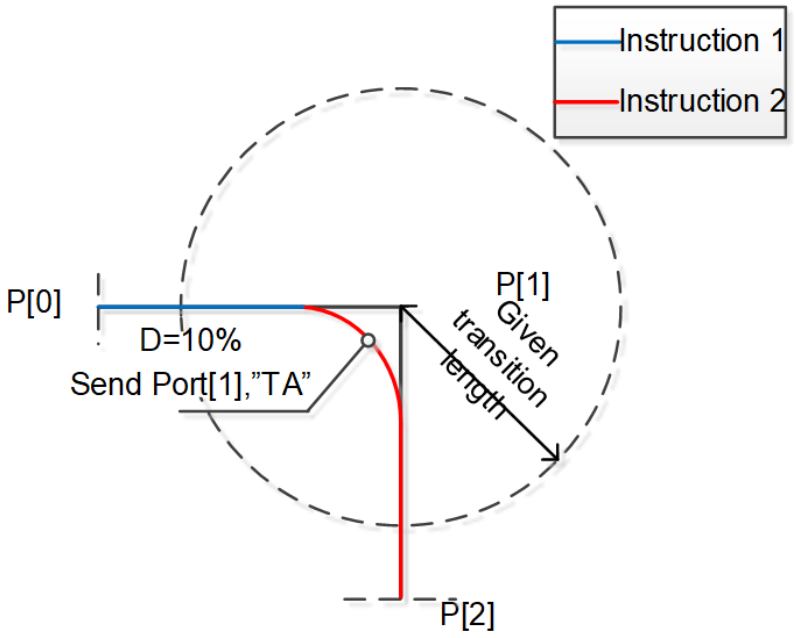
WaitInPos(10)##WaitInPos(Corner) can also be used

Send Port[1],"TA";

Analysis:

The robot starts transitioning before reaching P[1] and executes Send Port [1],"TA" after 10% of the MovJ P[2] motion is completed.

Then the robot stops at P[2].



7.14 Curve

- **Function**: Custom curve function and its call.
- **Description**: Curve is a custom function. It is mainly used to integrate multiple Movs instructions with different motion characteristics. It cannot be written into other instructions. Its call method in the main module is similar to the regular custom function.
- **Format**:

Custom curve parameter: Const CurveParm curve parameter name = {10,10,0,1,1,90,10,0,0};

Curve function name (List of parameters and default values)

…function body…

EndCurve;

- **Parameter list**:

CurveParm CurvePara=Custom default curve parameters, double VelRat=Custom default speed ratio, Int CurveTool=Custom default tool, Int CurveWobj=Custom default workobject.

1. **CurvePara**: Curve parameters defining the Curve function. If not specified, the default parameters are used. For details on the curve parameters, see the description of the free curve.
2. **V_Rate**: Speed percentage parameter defining the Curve function, 1 to 100. If not specified, the default parameters are used.
3. **CurveTool**: Curve tool parameters defining the Curve function. If not specified, the default parameters are used.
4. **CurveWobj**: Curve workobject parameters defining the Curve function. If not specified, the default parameters are used. When equal to -1, it indicates that the frame is dynamic and is used for the dynamic tracking process.

- **Description of function body**:

1. Only Movs motion instructions or comments can be added to the function body. Adding other motion instructions or non-motion instructions will trigger a compilation error.

2. The parameter setting of Movs instructions composing the curve is the same with that of the Movs instruction used independently.
3. When a Curve instruction is used, no variable can be used for any number in it.
4. The total number of points included in all Movs instructions in the Curve instruction ranges from 4 to 500.
5. The Tool and Wobj parameters in all Movs instructions of the curve do not take effect, and are subject to the default or actual passed values.
6. The curve uses the default curve parameters or the curve parameters passed during call. There is no need to add curve parameters to the Movs instructions.
7. The transition parameters between the curve and other instructions are subject to the transition parameters specified in the last Movs instruction of the curve.
8. The Nwait parameter between the curve and other instructions is subject to the Nwait parameter specified in the last Movs instruction of the curve.
9. In the curve, the actual reference speed of the Movs instruction is equal to the reference speed multiplied by the V_Rate percentage.

- **Function call format**:

1. Function call format of the current module: Function name (argument list)
2. Function call format of other modules: Module name.Function name (argument list); ##It is required to include the corresponding program name at the beginning of the program file.
3. The arguments may not be specified when the Curve function is called. In this case, the default values take effect.

- **Example**:

Main module ModelA.pro:

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

Const CurveParm C1 = {20,20,0,1,1,100,10,0,0};#Custom curve parameter, Const is a constant modifier, and the scope is the .pro file in which the parameter is included.

Curve myCurve(CurveParm CurvePara=C1, Double V_Rate=50, Int CurveTool=0, Int CurveWobj=0)

Movs P[147:200],V[30],Z[0],Tool[0],Wobj[0];

Movs P[200:300],V[30],Z[0],Tool[0],Wobj[0];

Movs P[300:447],V[30],Z[100],Tool[0],Wobj[0], Out(Out[5],ON,P[350]);

EndCurve;

Start;

#Calls functions in the same module

myCurve(C1,50,1,1);

myCurve( );

#Calls functions in another module

ModelB. curveA(C1,100,0,0);

ModelB. curveA( );

End;

Submodule ModelB.pro:

Const CurveParm C2={20,20,0,1,1,100,10,0,0};

Curve curveA CurveParm CurvePara=C1, Double V_Rate=50, Int CurveTool=0, Int CurveWobj=0)

Movs P[147:200],V[30],Z[0],Tool[0],Wobj[0];

Movs P[200:300],V[30],Z[0],Tool[0],Wobj[0];

Movs P[300:447],V[30],Z[100],Tool[0],Wobj[0], Out(Out[5],ON,P[350]);EndFunc;

EndCurve;

7.15 Return

- **Function**: Returns a value from a function.
- **Description**: Returns a value from the current function. The program returns to the main function without executing the subsequent instructions of the current function. The returned value is assigned to the left variable of the main function. Then the program continue to execute the subsequent statements.
- **Format**: Return variable/constant.
- **Supported return value type**:

1. Byte/Bool/Int/Float/Double/String type variables and constants.
2. B/LB/R/LR/D/LD/STR variables are supported.

- The null string is not supported.
- It is used with Func functions with return values.

7.16 Trap

- **Function**: Custom interrupt function and its call.
- **Description**: Callback function called by an interrupt.
- **Format**:

Custom function format:

Trap function name()

…function body…

EndTrap;

Function call format:

1. Current module function call format: IConnect IrqID,Function name().

2. Function call format for other modules: IConnect IrqID,Module name.Function name (); ##It is required to include the corresponding program name at the beginning 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 this module.

```
Trap funAdd()
  R[1] = aa + bb;
EndTrap;
Start;
#Calls functions in the same module
Int IrqID = 10;
IConnect IrqID, funAdd();
#Calls functions in another module
IConnect IrqID, ModelB. funDec();
End;
```

Submodule ModelB.pro:

```
Trap funDec()
  R[1] = aa - bb;
EndTrap;
```

8 Task Control Instructions

8.1 Xqt

- **Function**: Starts task.
- **Description**: This instruction is used to start an instruction-based PLC task, which allows multiple tasks to run together.
- **Format**: Xqt Index, filepath.
- **Parameter**:

Index: Task flag 3 (not task number). Temporarily retained for compatibility.

filepath: The program path name associated with the instruction-based task, such as "ABC.pro".

- **Example**:

Xqt 3, "ABC.pro";

8.2 Halt

- **Function**: Halts task.
- **Description**: Halts the selected task.
- **Format**: Halt Index.
- **Parameter**:

Index: Task flag 3 (not task number). Temporarily retained for compatibility.

- **Example**:

Halt 3;

8.3 Resume

- **Function**: Resumes task.
- **Description**: Resumes the selected task.
- **Format**: Resume Index.
- **Parameter**:

Index: Task flag 3 (not task number). Temporarily retained for compatibility.

- **Example**:

Resume 3;

8.4 Quit

- **Function**: Stops and quits task.
- **Description**: Stops and quits the selected task.
- **Format**: Quit Index
- **Parameter**:

Index: Task flag 3 (not task number). Temporarily retained for compatibility.

- **Example**:

Quit 3;

8.5 Pause

- **Function**: Stops operation.
- **Description**: Stops the operation. Equivalent to the function of the Stop button on the teach pendant. To resume the operation, press the start button. This instruction is often used to view variables during commissioning.
- **Format**: Pause.

- The Pause instruction in the static task stops the main task and the dynamic task.
- If motion exists (excluding independent axes) when this instruction is called, it waits for the motion to stop before taking effect.
- Calling the Pause instruction in an Xqt task will be invalid.

9 Operation Instructions

9.1 Variable Definition

- **Function**: Defines the variable.
- **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…};

Variable type Variable name[number of elements][number of elements] = {{element1, element2…}, {element1, element2…}};

Variable type 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.

- **Parameter**:

Variable type: Supports String, Bool, Byte, Int, Float, Double variables, or custom variables such as structs.

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 ternary array, the total array length cannot exceed 10000.

- You can initialize a variable when defining it. The length of a string variable for initialization is limited to 100 characters.
- The string variables support Chinese characters. However, it is important to note that the string operations in the instruction are based on single byte. The Chinese and English are encoded in different ways, as one Chinese character occupies two bytes. Therefore, you need to consider the difference between the two when performing string operations. For example, if you take a character from a Chinese string, the printed character will be illegible.

- **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.

9.2 Numerical Operations

9.2.1 Incr

- **Function**: Self-increment of numeric variables.
- **Format**:

Incr <Var>.

- **Parameter**:

<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

9.2.2 Decr

- **Function**: Self-decrement of numeric variables.
- **Format**:

Decr <Var>.

- **Parameter**:

<Var>: Integer variable, supports B/R/LB/LR/Int/Byte type variables.

- **Example**:

B[1]=1;

Decr B[1]; ##B1 decreased by 1, and the value changes to 0.

9.2.3 Sin

- **Function**: Sine operation.
- **Description**: Sine operation.
- **Format**: <Var>= Sin(<Exp>).
- **Parameter**:

<Exp>: Operation 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 perform forced conversion on the data.

- **Example**:

D[1]=Sin(60+D[2]);

9.2.4 Asin

- **Function**: Arc sine operation.
- **Description**: Arc sine operation.
- **Format**: <Var>= Asin(<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double, in angles.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= Asin (60+D[2])

9.2.5 Cos

- **Function**: Cosine operation.
- **Description**: Cosine operation, in angles.
- **Format**: <Var>= Cos(<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]=Cos(60+D[2]);

9.2.6 Acos

- **Function**: Arc cosine operation.
- **Description**: Arc cosine operation.
- **Format**: <Var>= Acos(<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double, in angles.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= Acos(60+D[2]);

9.2.7 Tan

- **Function**: Tangent operation.
- **Description**: Tangent operation, in angles.
- **Format**: <Var>= Tan(<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= Tan (60+D[2]).

9.2.8 Atan

- **Function**: Arc tangent operation.
- **Description**: Arc tangent operation.
- **Format**: <Var>= Atan (<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double, in angles.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= Atan (60+D[2]);

9.2.9 Sqrt

- **Function**: Square root operation.
- **Description**: Square root operation.
- **Format**: <Var>= Sqrt (<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

Example:

D[1]= Sqrt (9); ##D[1]=3

9.2.10 Pow

- **Function**: y-th power of x.
- **Description**: Calculates the y-th power of x.
- **Format**: <Var>= Pow(<Exp_x>, <Exp_y>).
- **Parameter**:

<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 perform forced conversion on the data.

- **Example**:

D[1]= Pow(2,3); ##D[1]=8

9.2.11 Abs

- **Function**: Absolute value.
- **Description**: Takes the absolute value.
- **Format**: <Var>= Abs (<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= Abs (-2); ##D[1]=2

9.2.12 Max

- **Function**: Compares values.
- **Description**: Takes the maximum of two values.
- **Format**: <Var>= Max (<Exp_x>, <Exp_y>).
- **Parameter**:

<Exp_x>, <Exp_y>: Expressions 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 perform forced conversion on the data.

- **Example**:

B[1]=1;

R[1]=2;

D[1]= Max(B[1],R[1]); ##D[1]=2

9.2.13 Min

- **Function**: Compares values.

- **Description**: Takes the minimum of two values.
- **Format**: <Var>= Min (<Exp_x>, <Exp_y>).
- **Parameter**:

<Exp_x>, <Exp_y>: Expressions 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 perform forced conversion on the data.

- **Example**:

B[1]=1;

R[1]=2;

D[1]= Min(B[1],R[1]); ##D[1]=1

9.2.14 AngleToRad

- **Function**: Converts an angle to a radian.
- **Description**: Converts an angle to a radian.
- **Format**: <Var>= AngleToRad (<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= AngleToRad (90)

9.2.15 RadToAngle

- **Function**: Converts a radian to an angle.
- **Description**: Converts a radian to an angle.
- **Format**: <Var>= RadToAngle (<Exp>).
- **Parameter**:

<Exp>: Operative expression.

Return value:

Return value type: double.

<Var>: Can be a numeric variable. If it is a non-double type variable, the system will perform forced conversion on the data.

- **Example**:

D[1]= RadToAngle (3.14)

9.3 String Operations

9.3.1 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).
- **Parameter**:

StrSource: Source string object.

n: Number of n 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".

9.3.2 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.

> NOTE
>
> n: Number of n characters (English).

- **Example**:

Str1 = "abcdefg";

Str2 = Right(Str1,2); ##Takes two right characters of Str1 and the result is Str2 = "fg".

9.3.3 Mid

- **Function**: Takes the middle characters of a string.
- **Description**: Takes the middle characters of the string, as shown in the following format, n characters starting from i^{th} character are taken.
- Format: StrDest = Mid(StrSource,i,n).
- **Parameter**:

StrSource: Source string object.

i: Start index, from 0.

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".

9.3.4 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).
- **Parameter**:

StrSource: Source string object.

i: Index of character to be taken, starting from 0.

Return value: StrDest: Truncated string.

- **Example**:

Str1 = "abcdefg";

Str2 = GetAt(Str1, 3); ##Takes the character indexed with 3 in Str1, and the result is Str2 = "d".

9.3.5 StrRePlace

- **Function**: String replacement.
- **Description**: Replaces the old substring OldSubStr with the new substring NewSubStr in a string.
- **Format**: StrDest = StrRePlace (StrSource,OldSubStr, NewSubStr).
- **Parameter**:

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".

9.3.6 StrReverse

- **Function**: Reverses string.
- **Description**: Reverses the order of characters in a string.
- **Format**: StrDest = StrReverse(StrSource).

- **Parameter**:

StrSource: Source string.

Rerurn value: StrDest: Reversed string.

- **Example**:

String Str1="aoe";

Str1= StrReverse (Str1); ##Result is Str1 = "eoa".

9.3.7 Strlen

- **Function**: Calculates the length of a string.
- **Description**: Calculates the length of a string.
- **Format**: length = Strlen(StrSource).
- **Parameter**:

StrSource: Source string.

Return value: length: String length.

- **Example**:

Str1 = "abcd";

LB[1] = Strlen(Str1); ##Result is LB1 = 4.

9.3.8 StrFind

- **Function**: String lookup.
- **Description**: Finds the index of a specified substring in a string.
- **Format**: Index = StrFind(StrSource,SubStr).
- **Parameter**:

StrSource: Source string.

SubStr: Substring to find.

Return value: Index: Specifies the starting index of the substring, starting from 0. -1 is returned when no result was found.

- **Example**:

Str1 = "abcde";

LR[1] = StrFind(Str1,"cd"); ##Result is LR1 = 2.

9.3.9 StrFindEnd

- **Function**: Reverse lookup.
- **Description**: Finds the last occurrence of the specified character in the string. The index of the character is returned if found successively, or -1 if failed.
- **Format**: Index = StrFindEnd (StrSource ,StrItem).
- **Parameter**:

StrSource: Source string object.

StrItem: Object to find in the source string.

Return value: Index: The index that was queried, starting from 0. -1 is returned when failed.

- **Example**:

LB[1]=StrFindEnd ("abcaba", "a"); ##Result is LB[1]=5.

9.3.10 TrimLeft

- **Function**: Trims left side of a string.
- **Description**: Removes the spaces to the left of the string.
- **Format**: StrDest = TrimLeft(StrSource).
- **Parameter**:

StrSource: Source string.

Return value: StrDest: Trimmed string.

- **Example**:

Str1 = " abc";

Str1 = TrimLeft(Str1);

##Result is Str1 = "abc"

9.3.11 TrimRight

- **Function**: Trims right side of a string.
- **Description**: Removes the spaces to the right of the string.
- **Format**: StrDest = TrimRight(StrSource).
- **Parameter**:

StrSource: Source string.

Return value: StrDest: Trimmed string.

- **Example**:

Str1 = "abc ";

Str1 = TrimRight (Str1);

##Result is Str1 = "abc"

9.3.12 Strcmp

- **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. When Str1 > Str2, a positive number is returned; when Str1 = Str2, 0 is returned; when Str1 < Str2, a negative number is returned.
- **Format**: value = Strcmp (Str1, Str2).
- **Parameter**:

Str1: String for comparison

Str2: String for comparison

Return value: value: Result of comparison. When Str1 > Str2, a positive number is returned; when Str1 = Str2, 0 is returned; when Str1 < Str2, a negative number is returned.

- **Example**:

Str1="ABxC";

Str2="ABzH";

R1=Strcmp(Str1,Str2); ##R1 is a positive number.

9.3.13 SPrintf

- **Function**: Formats a 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).
- **Parameter**:

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 (including float and double).
x(X): Hexadecimal integer.
s: String.

argument (Optional) A list of variables that can be any type of data. Depending on the Format parameter, the function expects a series of additional arguments, 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!"

9.4 Numeric Conversion

9.4.1 Caps

- **Function**: Converts a string to uppercase.
- **Description**: Converts a string to uppercase and return.
- **Format**: StrDest = Caps(StrSource).
- **Parameter**:

StrSource: Source string.

- **Return value**:

StrDest: Converted string.

Caps does not process non-English letters.

- **Example**:

Str1 = "aB12";

Str2 = Caps (Str1); ##Str2 = "AB12".

9.4.2 LowCase

- **Function**: Converts a string to lowercase.
- **Description**: Converts a string to lowercase and return.
- **Format**: StrDest =LowCase(StrSource).
- **Parameter**:

StrSource: Source string.

- **Return value**:

StrDest: Converted string.

LowCase does not process non-English letters.

- **Example**:

Str1 = "aB12";

Str3 = LowCase(Str1); ##Str 3= "ab12".

9.4.3 RToStr

- **Function**: Converts an integer variable to a string variable.
- **Description**: Converts an integer variable to a string variable.
- **Format**: StrValue = RToStr(Var).

- **Parameter**:

Var: Integer variable to be converted.

- **Example**:

R[1] = 45;

Str4 = RToStr(R[1]); ##Str4 = "45".

9.4.4 DToStr

- **Function**: Converts a double-type variable to a string variable.
- **Description**: Converts a double-type variable to a string variable.
- **Format**: StrValue = DToStr(Var,m,n).
- **Parameter**:

Var: Double-type 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.

- m defines the number of integers after conversion.
  - If m ≤ 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 ≤ 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.

9.4.5 StrToR

- **Function**: Converts a string to an integer data.
- **Description**: Converts a string to an integer data.
- **Format**: Var = StrToR(StrSource).
- **Parameter**:

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 variable

> NOTE
>
> - 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 "+", "-".
> - For 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.
> - The hexadecimal numbers support conversion of up to 64-bit data, that is, 8-bit hexadecimal numbers.
> - Binary numbers support conversion of up to 32-bit 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
```

9.4.6 StrToD

- **Function**: Converts a string to a double-precision data.
- **Description**: Converts a string to a double-precision data.
- **Format**: Var = StrToD(StrSource).

- **Parameter**:

StrSource: Format of 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.

NOTE

- 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", including 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 "+", "-".
- For 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 must not contain symbols such as "-", "、" and ".", otherwise the converted value will be incorrect.
- The hexadecimal numbers support conversion of up to 64-bit data, that is, 8-bit hexadecimal numbers.
- Binary numbers support conversion of up to 32-bit 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

9.4.7 PToStr

- **Function**: Converts point data to a string.
- **Description**: Converts point data to a string.
- **Format**: StrValue = PToStr(P).
- **Parameter**:

P: Point to be converted.

- **Return value**:

StrValue: Converted string

The first six coordinate values retain three decimal places after conversion.

- **Example**:

P[0]=(10,0,0,0,0,0),(1,0,0,0),(0,0,0,0,0,0);

Str[1]=PToStr(P[0]);

##The result of Str[1] is

10.000000,0.000000,0.000000,0.000000,0.000000,0.000000;1,0,0,0;0.000000,0.000000,0.000000,0.000000,0.000000,0.000000

9.4.8 BToAscii

- **Function**: Converts a Byte/B/LB type variable to corresponding characters by ASCII code.
- **Description**: Converts a Byte/B/LB type variable to corresponding characters by ASCII code.
- **Format**:

StrValue = BToAscii(Var).

- **Parameter**:

Var: Var: A Byte/B/LB type variable. If the input variable is a non-Byte type variable, it will be forcibly converted to a Byte type variable before the statement is executed.

- **Return value**:

StrValue: Converted string

- **Example**:

String NewStr;

LB[1]=65;

NewStr = BToAscii(LB[1]); ##Result is NewStr="A".

9.4.9 HexToStr

- **Function**: Function: Converts hexadecimal string.
- **Description**: Converts the hexadecimal string StrSource to the characters corresponding to the hexadecimal number, and store the string of characters in StrDest.
- **Format**:

length = HexToStr(StrSource,StrDest).

- **Parameter**:

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 HexToStr instruction to convert it to the string "!:qj4w" and store it in string StrDest.

9.4.10 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).

- **Parameter**:

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**:

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

9.4.11 GetStrAscii

- **Function**: Converts a string to ASCII code.
- **Description**: Converts a string to ASCII code and store the code in an array of B variables.
- **Format**: RetNum = GetStrAscii(StrSource,num,B).
- **Parameter**:

StrSource: Source string

num: The number of characters to convert to ASCII code. Range 0 to 255, cannot be greater than the length of the string.

B: B/LB variable, data is stored in a series of consecutive B variables starting from that B variable.

Return value:

RetNum: The number of characters successfully converted.

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

9.4.12 StrGetData

- **Function**: Gets string data.
- **Description**: Retrieves data from a string, stores it in the specified variable, and returns the number of data retrieved.
- **Format**: num = StrGetData(StrSource, Separator, Data).
- **Parameter**:

StrSource: Source string Currently, only local strings and Port are supported, and Port represents the string in the receive buffer.

Separator: The separator is a string.

Data: Data objects stored after splitting, 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 variables to split exceeds the range of the array, automatic truncation occurs.

- **Return value**:

num: The number of data retrieved

- **Example 1**:

String Str1 = "123And45Andggg";

B[0]= StrGetData(Str1,"And",R[0]);

The string is split into three strings: "123", "45", and "ggg", resulting in B[0]=3, R[0]=123, R[1]=45, and R[2]=0.

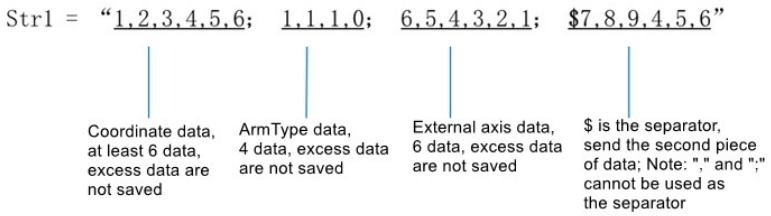
- **Example 2**:

String Str1 = "1,2,3,4,5,6; 1,1,1,0;6,5,4,3,2,1;$7,8,9,4,5,6;";

B[1]= StrGetData(Str1,"$",P[2]);

NOTE

- Description of Str1:



- The first split string "1,2,3,4,5,6;1,1,1,0;6,5,4,3,2,1" is stored in P[2], B[1]=2, as shown in the following figure:
  Robot:

P[002]			1.000	2.000	3.000	4.000	5.000	6.000
P[003]			7.000	8.000	9.000	4.000	5.000	6.000

  External axis:

P[002]			6.000	5.000	4.000	3.000	2.000	1.000
P[003]			0.000	0.000	0.000	0.000	0.000	0.000

  ArmTypes of P[2] is 1, 1, 1, and 0, respectively.
- For the second split string "7,8,9,4,5,6", the number of ArmType parameters and the number of external axis parameters do not meet the specification, but can be stored in P[3], with the missing ArmType and external axis parameters completed with 0.

9.4.13 CVDataToPose

- **Function**: Parses vision data into position variables.
- **Description**: Extracts data from string StrSource sent from vision equipment into position variable P.
- **Format**: CVDataToPose(StrSource, P, PVarMemset).
- **Parameter**:

StrSource: Incoming parameter, a source string object sent from vision equipment:

- **Format 1**: CVXY,X,Y,X,Y…, the frame head is CVXY, and the data will be parsed in the format of [X,Y].
- **Format 2**: CVXYA,X,Y,A,X,Y,A…, the frame head is CVXYA, and the data will be parsed in the format of [X,Y,A].
  - 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.

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;"

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;

11,21,0,0,0,0;1,0,0,0;

- **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**: CVDataToPose(StrSource, P, DVar,PVarMemset);
- **Parameter**:
  - 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: CVXYA,X,Y,A,X,Y,A…, the frame head is CVXYA, and the data will be parsed in the format of [X,Y,A].
  - Format: CVXYAL,X,Y,A,L,X,Y,A,L…, the frame header is CVXYAL, the data will be parsed in the format of [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.

NOTE

- 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.
- The L parameter indicates the type of object recognized by vision and is stored starting from the beginning of the DVar variable array.

Example:

P[0] = {(0,0,50,0,0,0),(1,0,0,0),(0,0,0,0,0,0)};

String StrSource = "CVXYAL,10,20,30,3,11,21,31,4";

R[0] = CVDataToPose(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;0,0,0,0,0,0;

11,21,50,31,0,0;1,0,0,0;0,0,0,0,0,0;

3

4

9.5 Coordinate Operations

9.5.1 Dist

- **Function**: Calculates the linear distance between two points.
- **Description**: Calculates the linear distance between two points.
- **Format**: Var = Dist(FromP, ToP).
- **Parameter**:

FromP: Start point.

ToP: Destination point.

Return value:

Return value type: double.

Var: Numeric variable that stores the returned value.

- Use the frame of the first position variable as a reference to determine the linear distance of the two points.
- Only position points are supported.

9.5.2 GetCurPos

- **Function**: Gets the current position point.
- **Description**: Gets the current position point.
- **Format**: P[*] = GetCurPos (Tool[*], Wobj[*]).
- **Parameter**:

Tool[*]: Optional parameter, defaults to the current tool of the system if omitted.

Wobj[*]: Optional parameter, defaults to the current workobject of the system if omitted.

It is the TCP of the current tool relative to the current workobject.

- **Example**:

P[1]= GetCurPos();

9.5.3 GetCurJPos

- **Function**: Gets the current joint point.
- **Description**: Gets the current joint point.
- **Format**: JP[*] = GetCurJPos().
- **Example**:

JP[1]= GetCurJPos()

9.5.4 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]),(E1,E2,E3,E4,E5,E6)}.
- **Parameter**:

(X,Y,Z,A,B,C): Pose values X,Y,Z,A,B,C.

(ArmType[0],ArmType[1],ArmType[2],ArmType[3]): Robot arm parameters, which vary depending on the robot model.

E1 to E6: Joint value of the external axis.

The arm parameters in the position variable is allowed to have a deviation of ±2 from the arm parameters corresponding to the actual reached position.

- **Example**:

P[3] = {(10,50,30,40,50,60),(1,0,0,0),(0,0,0,0,0,0)}

9.5.5 JP[i]=

- **Function**: Assigns a value to a joint point.
- **Description**: Assigns a value to a joint point.
- **Format**: JP[*]={(J1,J2,J3,J4,J5,J6),(E1,E2,E3,E4,E5,E6)}.
- **Parameter**:

J1 to J6: Joint value of the internal axis.

E1 to E6: Joint value of the external axis.

- **Example**:

JP[3] = {(10,50,30,40,50,60),(0,0,0,0,0,0)}

9.5.6 OffsetT

- **Function**: Offset along the current tool pose.
- **Description**: Offset along the current tool pose.
- **Format**:

OffsetT(P[***],PR[***])

OffsetT(P[***],X[*],Y[*]…)

- **Parameter**:

P[***]: Position variable subject to offset.

PR[***]: Offset variable.

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*] indicates the specified displacement in the X direction of the tool frame of the target point.

Note:

  - When the instruction OffsetT is used, offset is performed based on the current tool frame. PR is movement and rotation (X,Y,Z,A,B,C), offset sequence: X>Y>Z>A>B>C.
  - When used with motion instructions, such as MovJ OffsetT(P,PR), the motion instruction must contain Tool parameters, without Wobj parameters.

- **Example**:

PR[1]=(10,0,0,10,20,30);

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

Based on P[1], move along the current tool frame by 10 mm in the X direction, rotate 10° around Z, 20° around Y, and 30° around X.

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 target point can also be reached.

Movj OffsetT(P[1],X[10]) ,V[30], Z[0],Tool[1];

The end point of the tool is offset by 10mm under the frame of tool 1 on the basis of P[1].

9.5.7 Offset

- **Function**: Offset in Cartesian reference frame.
- **Description**: Offset of a target point in the Cartesian reference frame. The target point may be the TCP or flange center point. The reference frame is the frame of this target point.
- **Format**:

Offset (P[***],PR[***]);

Offset(P[***],X[*],Y[*]…);

- **Parameter**:

P[***]: Position variable subject to offset.

PR[***]: Offset variable.

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*] indicates the specified displacement in the X direction of the base frame of the target point.

NOTE

- 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 make an offset on coordinate values of position variables.

- **Example**:

P[1]={(300,100,-30,0,0,0),(1,0,0,0),(0,0,0,0,0,0)};

PR[1]=(10,0,0,0,0,0);

P[2]=Offset(P[1],PR[1]);

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

That is, P[2] is the result of offsetting P[1] by PR[1] in the frame where P[1] is located.

For Movj Offset (P,PR), the target point is offset in the frame where the target point is located.

9.5.8 Msft

- **Function**: Calculates offset.
- **Description**: Calculates the offset variable between two position variables. Only position points are supported.
- **Format**: PR = Msft (FromP,ToP).
- **Parameter**:

FromP: Start point.

ToP: Destination point.

Return value:

PR: PR/LPR variable, the calculation result.

CAUTION

Note: It is not allowed to obtain the offset variables when the two position points differ greatly in the arm parameters (For 6-axis robots, calculation of offset variables is not allowed when the difference in ArmType [0-2] exceeds 2. For floor-mounted and ceiling-mounted SCARA robots, calculation of offset variables is not allowed when the difference in ArmType [1-3] exceeds 2).

9.5.9 EOffsOn

- **Function**: Enables path offset.

- **Description**: Enables offset for the entire path. The final path of the motion will undergo additional offset along the X, Y and Z directions in the base frame. Once turned on, it remains active until the instruction EOffsOff disables the path offset.
- **Format**: EOffsOn(X,Y,Z).
- **Parameter**:

X: Offset in the X direction in the base frame.

Y: Offset in the Y direction in the base frame.

Z: Offset in the Z direction in the base frame.

- **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

- EoffsOn must be used in pairs with EoffsOff, and EoffsOn must be used before EoffsOff.
- The path offset applies only to instructions between EoffsOn and EoffsOff. The path offset is not related to the program logic, but only to the context. For example:
  Movj P[1],V[30],Z[0],Tool[1];

  B[1] = 2;

  EOffsOn(10, 0, 0);##The offset takes effect from this instruction, regardless of whether the instruction is executed.

  Movl P[2],V[30],Z[0],Tool[1];##Offset is applied to this instruction.

  If B[1]>1

  Goto L[1];

  EndIf;

  L[2]:

  Movl P[3],V[30],Z[0],Tool[1]; ##Regardless of where else the program jumps to this instruction, this instruction is subject to offset because it is located between EoffsOn and EoffSetoff.

  Func1();###This is a function call. The motion instructions in the function are not subject to offset.

  EOffsOff;

  L[1]:

  Movl P[4],V[30],Z[0],Tool[1];##Even if the program jumps to this instruction, this instruction is not subject to offset because it is outside EoffsOn and EoffSetoff.

  Goto L[2];
- Although the motion instruction with EOffsOn is independent of the program logic, the value of the PR variable in the EOffsOn instruction takes effect upon the execution of instruction. Therefore, do not add a conditional judgment before the EOffsOn instruction to prevent the offset variable from not 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(10,0,0);

  In the above program, the offset instruction takes effect, but the offset variable corresponding to the offset instruction does not. That is, the value of the PR variable is the previous one (0,0,0).

9.5.10 EOffsOff

- **Function**: Disables path offset.
- **Description**: Disables path offset. With EOffsOn, path offset is always enabled until EOffsOff is used.

- **Format**: EOffsOff.

For details, see 9.5.9 EOffsOn.

9.5.11 P=Offset

- **Format 1**:

DestP = OffSetT(SourceP,PR);

DestP = OffsetT (SourceP,X[*],Y[*]…);

- **Parameter**:

SourceP: Source point.

PR/LPR variable, the offset amount.

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*] indicates the specified displacement in the X direction of the frame of the target point.

- **Return value**:

DestP: Point after the offset.

- **Format 2**:

Offset indicates the offset of a target point in the Cartesian reference frame. The target point may be the TCP or flange center point. The reference frame is the frame of the this target point. PR is movement and rotation (X, Y, Z, A, B, C). Offset sequence: X > Y > Z > A > B > C.

- **Example 3**:

Get a point in the base frame and the flange moves in the base frame.

P[1]={(300,100,-30,0,0,0),(1,0,0,0),(0,0,0,0,0,0)};

PR1=(10,0,0,0,0,0);

P[2]=Offset(P[1],PR1);

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

Since P[1] is a point in the base frame, the center of the flange offsets relative to the base frame.

9.5.12 PR Sum

- **Function**: PR sum/difference.
- **Description**: Addition and subtraction of two offset variables.
- **Format 1**: PR[3] = PR[1] + PR[2].
- **Format 2**: PR[3] = PR[1] - PR[2].

> NOTE
>
> Directly performs numerical addition and subtraction operations on two offset variables and assigns values to another offset variable. It also applies to the local offset variable LPR.

- **Example**:

PR[3] = PR[1] - PR[2];

9.5.13 PR[i]=

- **Function**: Assigns a value to PR.
- **Description**: Directly assigns a value to the offset variable.
- **Format**: PR = (X,Y,Z,A,B,C).
- **Parameter**: (X,Y,Z,A,B,C): Offset parameters.
- **Return value**: PR: PR/LPR variable, object of assignment.

The PR variable only supports position points, that is, it represents offset variable in the Cartesian frame or the joint frame. For access to a single element of the offset variable: PR[*].X/Y/Z/A/B/C indicates the 1st to 6th elements of PR[*], which can represent the value of elements of an offset variable in the Cartesian frame or in the joint frame.

- **Example**:

PR[1]= (110,120,130,10,50,60); #Direct value 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]); ##Indirect value assignment using variables.

9.5.14 LoadPointFromFile

- **Function**: Loads point file.
- **Description**: Load points from the point file into the system. Note that only position points are supported.
- **Format**:

Int iRet = LoadPointFromFile(PtFile).

LoadPointFromFile(PtFile, Var).

- **Parameter**:

PtFile: Name of the point file.

iRet: Integer numeric variable that returns the total number of position variables in the point file.

Var: Integer numeric variable that returns the total number of position variables in the point file.

> **NOTE**
>
> Often used in conjunction with GetAPointFromFile instruction to get position variables from external point files.

The following is a common process for using the point files:

Example: Load point information from the point file "err.pt" and assign the first three points to P[0], P[1], and P[2] in sequence.

> **NOTE**
>
> - The point file uses ".pt" as a name suffix.
> - 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.
> - Every line is divided into three subparagraphs. The first six parameters compose the first paragraph and are coordinate values. The middle four parameters compose the second paragraph and are arm parameters. The last three parameters compose the third paragraph and are external axis values.
> - Paragraphs are separated from each other by ";", and the numbers within the paragraphs are separated by "," and end with ";".

You can specify 1 to 6 coordinate values, and replace the missing values with 0 if there are less than 6 values.

- **Format requirements**:

The coordinate values, arm parameters, and frame parameters are separated by ";".

The data in coordinate values, arm parameters, and frame parameters are divided with ",".

This instruction parses the required data in sequence (coordinate values, arm parameters, frame parameters). Excessive data is not parsed.

CAUTION

- The instruction does not have strict requirements on the format of the string. The string can be parsed as long as it is filled in according to the specification. If the data format does not conform to the common format, no error occurs. Ensure that the point data format is correct when using this instruction.
- In most cases, the point data input by this instruction is incomplete. This instruction does not check the validity of coordinate values. The validity of the data can only be checked during operation or calculation.

9.5.15 GetAPointFromFile

- **Function**: Gets points from point file.
- **Description**: Loads a single point into the position variable of the program from the system. Note that only the position point is supported.
- **Format**:

GetAPointFromFile(PtFile, Index , P).

P[*] = GetAPointFromFile(PtFile, Index).

- **Parameter**:

PtFile: Name of the point file.

Index: Index number (line number) of the point in the file.

P: The extracted point is placed in this position variable.

NOTE

See the description of 9.5.14 LoadPointFromFile.

9.5.16 WriteAPointToFile

- **Function**: PtFile: Point file name.
- **Description**: Writes points to a line in the specified point file. Note that only position points are supported.
- **Format**: WriteAPointToFile(PtFile, Index, P).
- **Parameter**:

PtFile: Name of the point file.

Index: Index number (line number) of the point to be stored in the file.

P: Position variable to be placed in the point file.

NOTE

After writing a point file, you must call the SavePointFile instruction to save it.

- **Example**:

WriteAPointToFile("eer.pt", 0, P[0]);

WriteAPointToFile("eer.pt", 1, P[1]);

SavePointFile ("eer.pt");

9.5.17 SavePointFile

- **Function**: Saves the point file.
- **Description**: Saves the point file. Note that only position points are supported.
- **Format**: SavePointFile (PtFile).
- **Parameter**: PtFile: Point file name

9.5.18 Lpallet

- **3-point method**
  Function: Defines a pallet by three points.

  Description: Sets the local pallet variable. It is often used in conjunction with P=Pallet.

  Three points P[i], P[j] and P[k] form a parallelogram, called the "boundary points" 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. The robot then can move to the specified position on the pallet according to the information of rows, columns, and layers.

  Format: LPallet LPalletNo,P[i],P[j],P[k],nRow, nColumn,nLay,LayHeight.

  Parameter:

  LPalletNo: Serial number of the local pallet variable, range 0 to 255.

  P[i], P[j], P[k]: Three points defining the pallet.

  nRow: Number of rows, range 1 to 1000.

  nColumn: Number of columns, range 1 to 1000.

  nLay: Number of layers, range 1 to 1000.

  LayHeight: Layer height, range 0 to 2000, in mm.

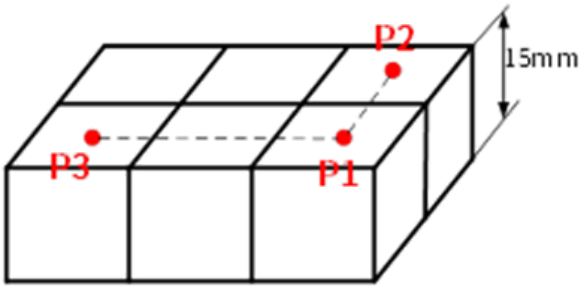
  

  ◦ You can define the pallet by a single row, column or layer, as well as by a point.

  ◦ You can define the pallet by a multiple rows, columns, or layers, not by a point.

  Example:

  LPallet 1,P[1],P[2],P[3],2,3,1,15;



- **4-Point method**

Function: Defines a pallet by four points.

Description: Sets the local pallet variable with four points. It is often used in conjunction with P=Pallet.

The four points P[i], P[j], P[k], and P[m] form a quadrilateral, called the "boundary points" 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 by lines and then 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]: Four points defining the pallet

nRow: Number of rows, range 1 to 1000.

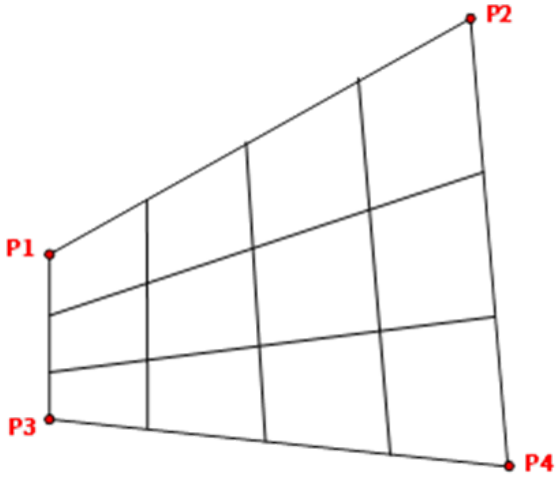
nColumn: Number of columns, range 1 to 1000.

nLay: Number of layers, range 1 to 1000.

LayHeight: Layer height, range 0 to 2000, in mm.

Example:

LPallet 2,P[1],P[2],P[3],P[4],5,4,1,15;



9.5.19 Pallet

- **3-point method**
  Function: Defines a pallet by three points.

Description: Sets the global pallet variable. It is often used in conjunction with P=Pallet. Three points P[i], P[j] and P[k] form a parallelogram, called the "boundary points" 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. The robot then can move to the specified position on the pallet according to the information of rows, columns, and layers.

Format: Pallet PalletNo,P[i],P[j],P[k], nRow, nColumn,nLay,LayHeight.

Parameter:

PalletNo: Serial number of the global pallet variable, range 0 to 255.

P[i], P[j], P[k]: Three points defining the pallet.

nRow: Number of rows, range 1 to 1000.

nColumn: Number of columns, range 1 to 1000.

nLay: Number of layers, range 1 to 1000.

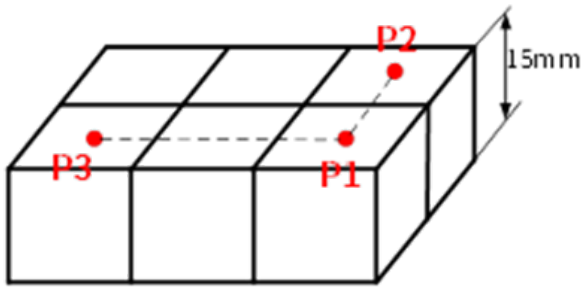
LayHeight: Layer height, range 0 to 2000, in mm.

CAUTION

- You can define the pallet by a single row, column or layer, as well as by a point.
- You can define the pallet by a multiple rows, columns, or layers, not by a point.

Example:

Pallet 1,P[1],P[2],P[3],2,3,1,15;



- 4-Point method

  Function: Defines a pallet by four points.

  Description: Sets the local pallet variable with four points. It is often used in conjunction with P=Pallet.

  The four points P[i], P[j], P[k], and P[m] form a quadrilateral, called the "boundary points" 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 by lines and then 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]: Four points defining the pallet

nRow: Number of rows, range 1 to 1000.

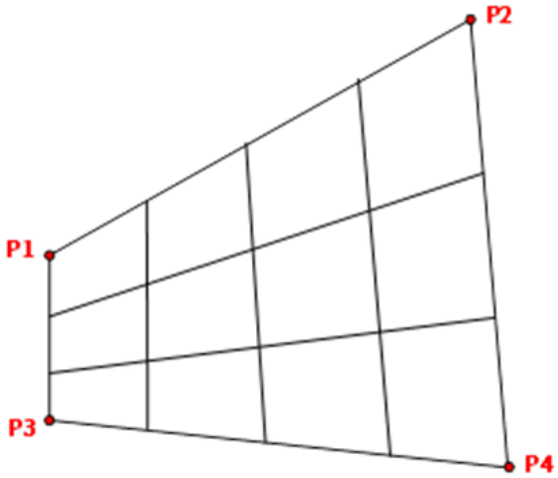
nColumn: Number of columns, range 1 to 1000.

nLay: Number of layers, range 1 to 1000.

LayHeight: Layer height, range 0 to 2000, in mm.

Example:

Pallet 2,P[1],P[2],P[3],P[4],5,4,1,15;



9.5.20 P=Pallet

- **Format 1**:

DestP = OffSetT(SourceP,PR);

DestP = OffsetT (SourceP,X[*],Y[*]…);

- **Parameter**:

SourceP: Source point.

PR/LPR variable, the offset amount.

X[*],Y[*],Z[*]: Select one or more, up to three of them. X[*] indicates the specified displacement in the X direction of the frame of the target point.

- **Return value**:

DestP: Point after the offset

- **Format 2**:

Offset indicates the offset of a target point in the Cartesian reference frame. The target point may be the TCP or flange center point. The reference frame is the frame of the this target point. PR is movement and rotation (X, Y, Z, A, B, C). Offset sequence: X > Y > Z > A > B > C.

- **Example**:

Get a point in the base frame and the flange moves in the base frame.

P[1]={(300,100,-30,0,0,0),(1,0,0,0),(0,0,0,0,0,0)};

PR1=(10,0,0,0,0,0);

P[2]=Offset(P[1],PR1);

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

Since P[1] is a point in the base frame, the center of the flange offsets relative to the base frame.

9.5.21 CalfixWobjPos

- **Function**: Converts world coordinates into non-grip fixed workobject coordinates.
- **Description**: Converts a position in the world frame into a position in the frame of a designated workobject that is fixed and not held by the robot (RobHold = False, and UFFix = True).
- **Format**: P[*] = CalfixWobjPos (WorldPos, Wobj[*]).
- **Parameter**:

WorldPos indicates a spatial point in the world frame.

Wobj[*] indicates the serial number of the fixed workobject.

P[*] indicates the value of this point in the workobject frame.

The arm parameter of P[*] is the same as that of WorldPos.

- **Example**:

P[1] = CalfixWobjPos (P[0], Wobj[1]);

9.5.22 CalRobholdWobjPos

- **Function**: Converts world coordinates into gripped workobject coordinates.
- **Description**: Converts a position in the world frame into a position in the frame of a designated workobject that is held by the robot (RobHold = True, and UFFix = True).
- **Format**: P[*] = CalRobholdWobjPos (WorldPos, Wobj[*], RefPos).
- **Parameter**:

WorldPos indicates a spatial point in the world frame.

Wobj[*] indicates the workobject held by the robot.

RefPos indicates the coordinate values of the flange in the world frame.

P[*] indicates the value of this point in the workobject frame.

The arm parameter of P[*] is the same as that of WorldPos.

- **Example**:

P[2] = CalRobholdWobjPos (P[0], Wobj[1], P[1]);

9.5.23 PToJ

- **Function**: Converts workobject coordinates into joint coordinates.
- **Description**: Converts the workobject coordinates under specified tool and workobject into the joint coordinates of the robot. You can specify the tool number and workobject number.
- **Format**: JP[*] = PToJ (P[*], Tool[*],Wobj[*]).
- **Parameter**:

P[*]: Coordinates of the workobject.

Tool[*]: Specifies the tool number.

Wobj[*]: Specifies the workobject number.

- **Return value**: Joint coordinates, JPos type.
- **Example**:

JP[1] = PToJ (P[1], Tool[1],Wobj[1]);

9.5.24 JToP

- **Function**: Converts joint coordinates into workobject coordinates.
- **Description**: Converts the joint coordinates into the workobject coordinates. You can specify the tool number and workobject number.
- **Format**: P[*] = JToP (JP[*], Tool[*],Wobj[*]);
- **Parameter**:

JP[*]: Joint coordinates of the robot.

Tool[*]: Specifies the tool number.

Wobj[*]: Specifies the workobject number.

- **Return value**: Workobject coordinates, RobPos type.
- **Example**:

P[1] = JToP (JP[1], Tool[1],Wobj[1]);

9.5.25 CVToRobPos

- **Function**: Converts vision coordinates into robot coordinates.
- **Description**: Converts the pixel coordinates sent from the vision equipment into the corresponding workobject coordinates.
- **Format**:

1. CVToRobPos(P[*], int VisionCrdNo);
   Meaning: Converts pixel coordinates into workobject coordinates when the camera is installed external to the robot.
2. CVToRobPos(P[*], int VisionCrdNo, JP[*]);

   Meaning: Converts the pixel coordinates into workobject coordinates when the camera is installed on the robot arm. JP is the joint coordinates of the photo-taking point.

   Parameter:

P[*]: Pixel coordinates sent from the vision equipment, in which the arm parameter and external axis data use the arm parameter and external axis data of the target point.

VisionCrdNo: Vision frame number.

JP[*]: Joint coordinates of the photo-taking point when the camera is mounted on the robot arm.

- **Example**:

P[0] = {(10,20,0,0,0,0),(1,0,0,0),(0,0,0,0,0,0)};

P[1] = CVToRobPos(P[0],1,JP[0]);

NOTE

- The coordinates obtained after CVToRobPos conversion represent a point in the base frame, independent of the set workobject frame.
- In addition, the CVToRobPos conversion only involves X, Y and A data in the point. The Z, B and C data obtained after conversion are independent of the Z, B and C data in the point before conversion.

9.5.26 SavePoints

- **Function**: Saves the currently loaded global point file.
- **Description**: Saves the modified coordinates of the global position points in the program to the currently loaded global point file.
- **Format**: SavePoints.
- **Parameter**: /.
- **Example**:

P[0].X = 10;

P[1].Y = 3.14;

P[3].Z = -10;

SavePoints;

CAUTION

- If the LoadPoints instruction is used in the program, ensure that the currently loaded global point file is the one you want to save to prevent the point file from being overwritten by mistake.
- If the LoadPoints instruction is used before the SavePoints instruction, all the point changes made before the LoadPoints instruction will be reset. Design the program logic with caution.
- After the SavePoints instruction is called, the contents of the global point file displayed on the teach pendant interface will not be updated immediately. To update the global point file, click "Sync to local" button in the InoRobotLab software. Also, you need to click the "Refresh" button on the project screen in the InoTeachPad software and the handheld teach pendant.
- The frequent use of the SavePoints instruction will reduce the efficiency of task execution and affect the service life of the storage medium of the robot controller. It is recommended to use the SavePoints instruction only when point data needs to be saved. You should avoid using the SavePoints instruction simultaneously for multiple tasks.
- If a decoding error occurs when the SavePoints instruction is used in a static task, the current static task will be stopped. You need to resynchronize the project to resume the task.

10 Communication Instructions

10.1 Open Socket

- **Function**: Connects the robot as a client to an external server.
- **Description**: Specifies the server IP address and port number for connection to the server.
- **Format 1**: Open Socket(IP,SvrPort,ClientPort, BufType).
- **Format 2**: Open Socket(IP,SvrPort,ClientPort, BufType,B).
- **Parameter**:

IP: Server IP address

SvrPort: Server port number, 1 to 65535.

ClientPort: Client port number, 2 to 17, or 1024 to 65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

BufType: The way to store data in the buffer (capacity: 1024 bytes, up to 1023 bytes of valid data including the terminator '\0' can be transmitted), supporting circle and single. In "single" mode, the robot can receive a maximum of 1023 bytes of valid data at a time, and will discard data exceeding 1023 bytes. In "circle" mode, 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 GetAString instruction, the data in the buffer will be accumulated till the buffer is full.

Single: When the data is obtained by Get Port or GetAString instruction, only the data received at that time will be acquired.

B: (Optional) Return value (1 for success, 0 for failure), only B/LB variables are supported.

NOTE

- The Open Socket instruction is used to enable local controller as a client to communicate with external server.
- Once the Open Socket instruction takes effect, the port remains open. You can close the port using the Close Socket instruction. The port will be automatically closed when another program is opened manually.
- The instruction is looped. If it is determined that the port has been opened based on the return value, the program jumps out of the loop.
- When the server is not turned on, if the instruction of Format 1 is used, the function will get stuck until the server is turned on.
- If the instruction of Format 2 is used, the function runs normally and you can tell if the connection is successful by the return value B/LB and the prompt in the notification bar.
- When connection is made with specified port number (1024-65535), after the communication is disconnected by the Close instruction, 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 Open Socket instruction to establish a new connection.

- Note:

Restrictions on port number: When the robot acts as a client, the port number ranges from 1024 to 65535. 1111, 2222, 3333, 4444, 5555, 6666, 7777, 8888, 9999, 1217, 8080, 11740, and 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	Generally, the communication over these ports clearly indicates the protocol for a service.
1024 to 65535 Registered ports	1111/7777/8888/9999	Reserved for the robot system.
	1217	CODESYS application port.
	2222	API only.
	3333	Teach pendant only.
	4444	When a local controller acts as the server and an external device acts as the only client (the index of the first connection is #2) whose port number is unknown, you can use 4444.
	5555	Tooling test only.

Port number distribution	Port number	Application
	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;

10.2 Open Com

- **Function**: Connects to serial port.
- **Description**: Specifies the server IP address and port number for connection to the serial port.
- **Format 1**: Open Com(Port, Baudrate).
- **Format 2**: Open Com(Port, Baudrate,B).
- **Parameter**:

Port: Serial port number (The controller has only one serial port and the serial port number is 1.)

Baud rate: 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200

B: (Optional) Return value (1 for success, 0 for failure), only B/LB variables are supported.

Once the Open Com instruction takes effect, the port remains open. You can close the port using the Close Socket instruction. The port will be automatically closed when another program is opened manually.

CAUTION

- Before using this instruction, ensure that the user serial port function is enabled. (The user serial port function is disabled by default.)

There are two ways to enable the user serial port function:

Method 1: Configure the controller through the teach pendant.

On the connected teach pendant, go to "settings" > "System settings" > "TP Port Setting" and enable the serial port. When finished, restart the controller.

Method 2: Configure the controller through SecureCRT software.

- After connecting to the controller through the SecureCRT software, enter the account: root, password: r.
- Then enter the start command: Set_serial 1.
- Restart the controller.

- When the serial port is cyclically opened and closed, please add 4 to 5s interval after the Open Com instruction.

- **Example**:

```
LB[1] = 0;
While  LB[1]<>1
Open Com(1, 9600, LB[1]);
EndWhile;
```

10.3 Close Socket

- **Function**: Closes the Ethernet port.
- **Description**: Closes the open Ethernet port to close the Ethernet connection.
- **Format**: Close Socket,ClientPort.
- **Parameter**: ClientPort: Client port to be closed.
- **Range**: 2 to 17, or 1024 to 65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

NOTE

- The Close Socket instruction is used to close the open Ethernet port. It can be used in the following two scenarios:
  - Close the connection between the local controller and the external device when the controller acts as a client.
  - Close the connection between the local controller and the external device when the controller acts as a server.

- **Example**:

Close Socket, 1025;

10.4 Close Com

- **Function**: Closes the serial port.
- **Description**: Closes the open serial port to close the serial port connection.
- **Format**: Close Com, Port.
- **Parameter**: Port: Serial port number. (The controller has only one serial port and the serial port number is 1.)

CAUTION

When the serial port is cyclically opened and closed, add 4 to 5s interval after the Close Com instruction.

- **Example**:

Close Com,1;

10.5 Send Port

- **Description**: Sends a string to a remote port on the network.
- **Format 1**: Send Port[PortNo],StringValue.
- **Parameter**:

PortNo: In client-server communication, it is the client port number.

- **Range**: 1 to 17, or 1024 to 65535.

When the serial port is used for communication, this PortNo is the serial port number (only one serial port is available and the serial port number is 1). 2 to 17 indicate the 1st to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

StringValue: The string sent.

- **Format 2**: Send Port[PortNo],StringValue,Type.
- **Parameter**:

PortNo: In client-server communication, it is the client port number. When the serial port is used for communication, this PortNo is the serial port number (only one serial port is available and the serial port number is 1).

StringValue: The string sent.

Type: Type of data sent, can be String/Binary/Hex. When this parameter is not present, the data is sent as a string by default. Description:

String: The data is sent as a string.

Binary: Converts a binary string to a corresponding binary value for sending. (Temporarily invalid)

Hex: Converts a hexadecimal string to a corresponding hexadecimal value for sending.

- **Format 3**: Send Port[PortNo], StringValue, OnceAndErrNoStop, B/LB.
- **Parameter**:

PortNo: In client-server communication, it is the client port number.

- **Range**: 1 to 17, or 1024 to 65535.

When the serial port is used for communication, this PortNo is the serial port number (only one serial port is available and the serial port number is 1). 2 to 17 indicate the 1st to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

StringValue: The string sent.

OnceAndErrNoStop: (Optional) When this parameter is selected, the process will not block upon data transmission failure and will continue execution. When not selected, the original specification is maintained.

B/LB: (Optional) Return value (1 for success, 0 for failure), only B/LB variables are supported.

- **Format 4**: Send Port[PortNo], StringValue, Type, OnceAndErrNoStop, B/LB.
- **Parameter**:

PortNo: In client-server communication, it is the client port number. When the serial port is used for communication, this PortNo is the serial port number (only one serial port is available and the serial port number is 1).

StringValue: The string sent.

Type: Type of data sent, can be String/Binary/Hex. When this parameter is not present, the data is sent as a string by default. Description:

String: The data is sent as a string.

Binary: Converts a binary string to a corresponding binary value for sending. (Temporarily invalid).

Hex: Converts a hexadecimal string to a corresponding hexadecimal value for sending.

OnceAndErrNoStop: (Optional) When this parameter is selected, the process will not block upon data transmission failure and will continue execution. When not selected, the original specification is maintained.

B/LB: (Optional) Return value (1 for success, 0 for failure), only B/LB variables are supported.

NOTE

- In client-server communication, the local controller can acts as a client and a server. However, the port number here is always the client port number.
- When a local controller acts as the server and an external device acts as the only client (the index of the first connection is #2) whose port number is unknown, you can use 4444.
- When the serial port is used for communication, this PortNo is the serial port number.

After the system is rooted, the user serial port function is disabled by default.

To open the user serial port function:

1. After connecting to the controller through the SecureCRT software, enter the account: root, password: r.
2. Then enter the start command: Set_serial 1.
3. Restart the controller, or in the InoRobotLab software, go to "Controller parameter configuration" > "Debug" > "COM switch" to open the serial port.
4. In TCP/serial communication, you cannot send or receive data to or from the same port/serial port at the same time in multiple tasks. Otherwise, communication errors will occur.
5. B/LB parameters need to be used with the OnceAndErrNoStop parameter, otherwise they are invalid.
6. In "Client" mode, the ComType optional parameter can only be ClientType. In "Server" mode, the ComType optional parameter can only be ServerType. Otherwise, a runtime error will occur. In both modes, using the keyword ComType has the same functionality as not using it.

- **Example**:

```
##Send a string "ABC"
Send Port[1025],"ABC",String;
##Send a global string variable
Str[1] = "ABC";
Send Port[1025],Str[1] ,String;
##Send a local string variable
String ss = "ABC";
Send Port[1025],ss,String;

##Send hexadecimal data
Str[1] = "213A716A3477";
length = StrToHex(Str[1], Str[2]);
Send Port[1025],Str[2],Hex;
```

10.6 Get Port

- **Function**: Receives data from the remote port.
- **Description**: Within a certain period of time, receive data from the remote port and place it in a buffer. When no data is received within a specified time, jump to a specified tag.

- **Format**: Get Port[PortNo],T[Time],Goto L[Index].
- **Parameter**:

PortNo: In client-server communication, it is the client port number.

- **Range**: 1 to 17, or 1024 to 65535.

When the serial port is used for communication, this PortNo is the serial port number (only one serial port is available and the serial port number is 1). 2 to 17 indicate the 1st to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

Time: Wait time, 0 to 65535, in second

Index: The index of label to which the program jumps upon wait time out.

NOTE

- In client-server communication, the local controller can acts as a client and a server. However, the port number here is always the client port number.
- When a local controller acts as the server and an external device acts as the only client (the index of the first connection is #2) whose port number is unknown, you can use 4444.
- When the serial port is used for communication, this PortNo is the serial port number. After the system is rooted, the user serial port function is disabled by default.

- **To open the user serial port function**:

Method 1:

1. After connecting to the controller through the SecureCRT software, enter the account: root, password: r.
2. Then enter the start command: Set_serial 1.
3. Restart the controller.

Method 2:

1. In the InoRobotLab software, go to "Controller parameter configuration" > "Debug" > "COM switch" to open the serial port.
2. Data is received via port within T[Time] time. If the reception is successful, the data is saved in the receiving buffer and the next line of instruction continues to be executed (without executing Goto L[Index]). If no data is received, GotoL[Index] is executed and the program jumps to L[Index].
3. When BufType in the Open Socket instruction is set to "circle", the Get Port instruction will get all the data since the last Get Port instruction; when set to "single", the Get Port instruction will get the most recently received data. Buffered data will be automatically cleared after new data is acquired.

- **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".

10.7 GetPortbuf

- **Function**: Gets string from receive buffer.
- **Function**: 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).
- **Parameter**:

StartBit: Starting read position of the receive buffer, range 0 to 1023.

Num: Number of characters read, range 1 to 100; for reception via the serial port, range 1 to 16.

Port: In client-server communication, this Port is the client port number.

- **Range**: 1 to 17, or 1024 to 65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

When the serial port is used for communication, this Port is the serial port number. The controller has only one serial port and the serial port number is 1.

- **Return value**: StrVar: Saves the string read from the receive buffer.

> NOTE
>
> - A specified number (Num) of characters is retrieved from a StartBit in the receive buffer and passed to the specified string variable. 0 is passed upon failure.
> - The GetPortbuf instruction must be used in conjunction with the Get Port instruction. Otherwise, the obtained string is empty, resulting in subsequent logic errors.

- **Example**:

##Get three characters from the second bit

L[2]:

Get Port[1025],T[1],Goto L[2];

Str1 = GetPortbuf(2,3,1025);

10.8 GetPortState

- **Function**: Gets the status of the specified socket connection.
- **Description**: Gets the status of the specified socket connection.
- **Format**: GetPortState(PortNo).
- **Parameter**:

PortNo: Client port number, range 1 to 17, or 1024 to 65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

When the serial port is used for communication, this PortNo is the serial port number. The controller has only one serial port and the serial port number is 1.

- **Return value**:

1: Connection is normal.

0: Connection is abnormal.

> NOTE
>
> - When using GetPortState, it is recommended to add a 1 to 2s delay before the instruction to wait for the network status to stabilize.

10.9 GetSocketNo

- **Function**: Gets the client port number.
- **Description**: Gets the client port number for the specified communication.
- **Format**: GetSocketNo("IPAdress",RVar).

- **Parameter**:

IPAdress: IP address of the external device In particular, when the IP address is 0.0.0.0, all client port numbers are obtained.

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, the port numbers of all connections on the client are stored in an array of variables starting with the R/LR variable.

- **Return value**:

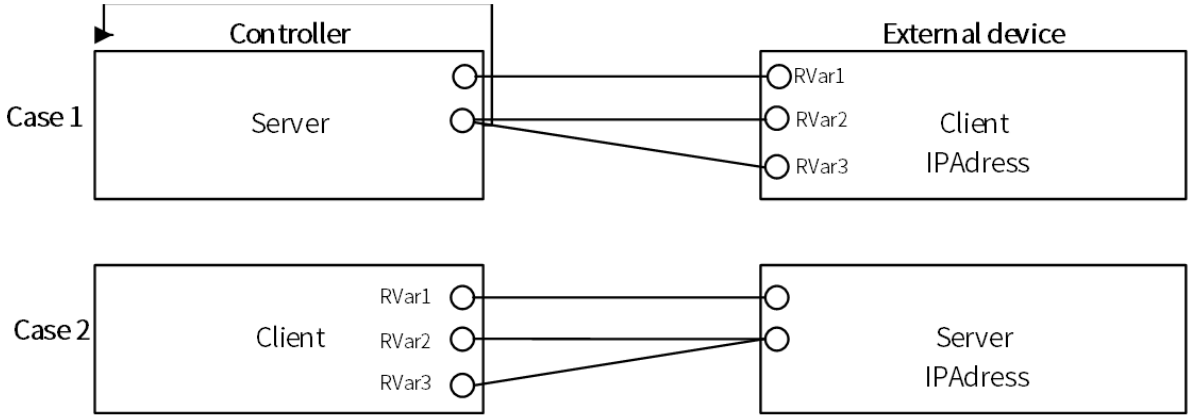
Returns the number of connected clients.

NOTE

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.

CAUTION

- What is obtained is the port number of the client.
- When the controller has multiple connections to external devices, the client port numbers of all these connections are obtained.



- The GetSocketNo instruction gets the general communication port number (general communication is established through "Settings" > "System settings" > "Communication settings" > "Communication manager" on teach pendant, or through the Open Socket instruction). It cannot get the special communication port number (such as communication established through "Settings" > "Vision Calibration").
- In "Client and Server" mode, GetSocketNo prioritizes obtaining the communication port number where the controller acts as the client, followed by that where the controller acts as the server.

- **Example**:

B[1] = GetSocketNo("192.168.23.100",R[3]);

If(B[1]>0)

##Gets the first connection

Get Port[R[3]],T[1],Goto L[1];##Gets the data

Endif;

10.10 SetSocketRecvType

- **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, a bnary or hexadecimal string. (Binary string is temporarily invalid.)
- **Format**: SetSocketRecvType(Port,Type).
- **Parameter**:

Port: 1 to 17, or 1024-65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

When the serial port is used for communication, this Port is the serial port number. The controller has only one serial port and the serial port number is 1.

Type: Type of data received, which can be String/Binary/Hex. Description:

String: Receives data in a string form.

Binary: Receives data in binary form and converts it to a binary string (temporarily invalid).

Hex: Receives data in hexadecimal form and converts it to a hexadecimal string.

(You can also use 0 for String, 1 for Binary, and 2 for Hex.)

NOTE

- In client-server communication, the local controller can acts as a client and a server. However, the port number here is always the client port number.
- When a local controller acts as the server and an external device acts as the only client whose port number is unknown, you can use 4444.
- The scope of the instruction is the entire system, including the main program, subprogram, multi-threaded. The instruction is initialized to the String type when the program exits.
- To switch the receive data type, it is recommended to use the SetSocketRecvType instruction after the Open Socket instruction.
- The SetSocketRecvType instruction must be executed before the peer sends data, so that the set receive type Hex or String can take effect.

- **Example**:

##When an external device sends data in hexadecimal, the robot can receive in the following two ways:

```
SetSocketRecvType(1025, Hex);
Get Port[1025],T[1],Goto L[2];
Str2 = GetPortbuf(2,3,1025);
```

```
Get Port[1025],T[1],Goto L[2];
Str1 = GetPortbuf(2,3,1025);
StrToHex(Str1,Str2);
```

10.11 GetAString

- **Function**: Receives data from the 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].
- **Parameter**:

Port: In client-server communication, this Port is the client port number.

- **Range**: 1 to 17, or 1024 to 65535.

2 to 17 indicate the first to 16th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

When the serial port is used for communication, this Port is the serial port number (only one serial port is available and the serial port number is 1).

Time: Wait time, 0 to 65535, in second

Str: Split string.

Sep: Separator.

Index: The index of label to which the program jumps upon wait time out.

NOTE

- In client-server communication, the local controller can acts as a client and a server. However, the port number here is always the client port number.
- When a local controller acts as the server and an external device acts as the only client whose port number is unknown, you can use 4444.
- When the serial port is used for communication, this Port is the serial port number. After the system is rooted, the user serial port function is disabled by default. To open the user serial port function:
  1. After connecting to the controller through the SecureCRT software, enter the account: root, password: r.
  2. Then enter the start command: Set_serial 1.
  3. Restart the controller.
- Data is received via port within T[Time] time. If the reception is successful, the data is saved in the receiving buffer and the next line of instruction continues to be executed (without executing Goto L[Index]). If no data is received, GotoL[Index] is executed and the program jumps to L[Index].
- If BufType in the Socket instruction is set to "circle", the GetAString instruction gets all the data separated by a separator between the last reception and this reception. The data successfully obtained is deleted from the cache queue and the subsequently received data is added to the queue. If set to "single", the GetAString instruction gets the latest received data and clears the previously received data.

- **Example**:

GetAString Port[1024],T[1],Str[1],"\n",Goto L[2];

Application 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 cached 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];

- When data is retrieved from the cached data queue, the data retrieved includes the separator. For example, in the above application case, Str[1] ends with "\n".
- The separators "\n" and "\r" are specially processed, while all other characters are parsed according to the actual number of characters.

10.12 GetDynamicIP

- **Function**: Gets dynamic IP (internally uses EtherNet/IP).
- **Format**: <Exp_i>=GetDynamicIP(<StrSouce>).
- **Parameter**: <StrSouce>, a string that stores the current dynamic IP address.
- Str type and String type variables available:

Str[i]: i ranges from 0 to 255.

String: Custom name.

- **Return value**: <Exp_i>, acquisition status returned.

(1 for failure, 0 for success).

B, LB, R, LR, D, LD variables available:

B[i], LB[i], R[i], LR[i], D[i], LD[i]: i ranges from 0 to 255.

NOTE

- General scenario: When the dynamic IP is unknown, you need to view the current dynamic IP setting through the GetDynamicIP instruction.
- Key scenario: When multiple robots are managed by one industrial computer, project synchronization is prone to errors and there is a risk of robot collision. With the dynamic IP acquisition instruction, the dynamic IP is acquired for comparison with the preset IP. In case of mismatch, an error occurs.

- **Example 1**: Standard type

```
Str[0] = "";
B[0] = GetDynamicIP(Str[0]);
If B[0] == 0
Print Str[0];
Else
Print "Failed to get dynamic IP"
```

Example 2: Custom type

```
String vanIP;
INT result;
result = GetDynamicIP(vanIP);
If result == 0
Print (vanIP);
Else
Print "Failed to get dynamic IP"
```

Application case:

In the program editor of the teach pendant or the InoRobotLab software, use this instruction to view the current IP address.

11 Information Interaction Instructions

11.1 Alarm

- **Function**: Displays custom alarms.
- **Description**: Displays user-defined alarm information in the message bar.
- **Format 1**: Alarm [Index].
- **Parameter**: Index: Custom alarm number (0 to 15).
- **Example**:

Alarm [2].

- **Format 2**: Alarm [Index],EasyGo.
- **Parameter 1**: Index: Custom alarm number (0 to 15).
- **Parameter 2**: EasyGo: When a user-defined alarm occurs, the program automatically moves to the next line. After the alarm is cleared, the program starts from the next line.

NOTE

- If this instruction is called in the main task or dynamic task, an alarm occurs. The main task and all dynamic tasks stop. The program of the task which has triggered Alarm[id],Easygo moves to the next line. When the alarm is cleared and the program is restarted, the program runs from the next line.
- If this instruction is called in a static task, an alarm occurs. The main task, all dynamic tasks, and the static task which has triggered Alarm[id],Easygo stop. The program of the static task which has triggered Alarm[id],Easygo moves to the next line. When the alarm is cleared and the static task is activated again, the program runs from the next line. If the static task is activated again without clearing the alarm, the static task will also run from the next line.

- **Example**:

Alarm [2],EasyGo;

11.2 Print

- **Function**: Prints information.
- **Description**: Prints variables or strings in the message bar.
- **Format**: Print[Object].
- **Parameter**: Object: Variable or string to be printed.

Due to screen size limit of the teach pendant, a maximum of five expressions can be connected for printing.

NOTE

- The printed content cannot exceed 99 characters (up to 33 Chinese characters).
- When the two Print instructions are used consecutively, in order to facilitate debugging, a delay (typically 0.5s) can be added to facilitate commissioning.

- **Example**:

Print B[1]+P[2]+LR[1]

11.3 GetPlcVarByte

- **Function**: Gets the value of a variable of Byte type.
- **Description**: Gets the value of a Byte-type variable from the PLC.
- **Format**: GetPlcVarByte(VarIndex).
- **Parameter**: VarIndex: Index of the PLC's Byte-type variable (0 to 255).
- **Return value**: Read value of the PLC's Byte-type variable, return value type: Byte.
- **Example**:

B[1]= GetPlcVarByte(3);

11.4 GetPlcVarInt

- **Function**: Gets the value of a variable of Int type.
- **Description**: Gets the value of an Int-type variable from the PLC and stores it.
- **Format**: GetPlcVarInt(VarIndex).
- **Parameter**: VarIndex: Index of the PLC's Int-type variable (0 to 255).
- **Return value**: Read value of the PLC's Int-type variable, return value type: Int.
- **Example**:

R[1]= GetPlcVarInt(3);

11.5 GetPlcVarDInt

- **Function**: Gets the value of a variable of DInt type.
- **Description**: Gets the value of a DInt-type variable from the PLC.
- **Format**: GetPlcVarDInt(VarIndex).
- **Parameter**: VarIndex: Index of the PLC's DInt-type variable (0 to 255).
- **Return value**: Read value of the PLC's DInt-type variable, return value type: Int.
- **Example**:

R[1]= GetPlcVarDInt(3)

11.6 GetPlcVarLReal

- **Function**: Gets the value of a variable of LReal type.
- **Description**: Gets the value of a LReal-type variable from the PLC.
- **Format**: GetPlcVarLReal(VarIndex).
- **Parameter**: VarIndex: Index of the PLC's LReal-type variable (0 to 255).
- **Return value**: Read value of the PLC's LReal-type variable, return value type: double.
- **Example**:

D[1]= GetPlcVarLReal(3);

11.7 TimeStart

- **Function**: Starts timer.
- **Description**: Starts timer for timing.
- **Format**: TimeStart(TimerIndex).
- **Parameter**: TimerIndex: Index of the timer (0 to 99).
- **Example**:

TimeStart(0);

> CAUTION
>
> - After the timer is started, the value accumulates until the timer is started again using the TimeStart instruction.
> - The timer runs in the controller and will not be restarted due to program switching, unless it is restarted using the TimeStart instruction.

11.8 TimeOut

- **Function**: Outputs timer duration.
- **Description**: Outputs timer duration to a specified variable.
- **Format**: TimeOut(TimerIndex, Var).
- **Parameter**: TimerIndex: Index of the timer (0 to 99).

Var: Double-type variable, storing the timer duration (s).

> CAUTION
>
> A timer that is not started can be compiled, but the value is invalid.

- **Example**:

START;

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

##Start Timer 1

TimeStart(1);

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

##Reads Timer 1

Timeout(1,D[2]);

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

##Reads Timer 1

Timeout(1,D[3]);

END;

11.9 GetModBusCoil

- **Function**: Gets coil values from ModBus slave.
- **Description**: Gets coil values from ModBus slave and saves them in a Byte-type variable.
- **Format**: GetModBusCoil(StartAddr, CoilNum, Var).
- **Parameter**: StartAddr: Start address of the coil (0 to 8191), in bit.

CoilNum: The number of coils to read (1 to 8)

Var: Byte/B/LB-type variable, used to store the read coil values. 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.

CAUTION

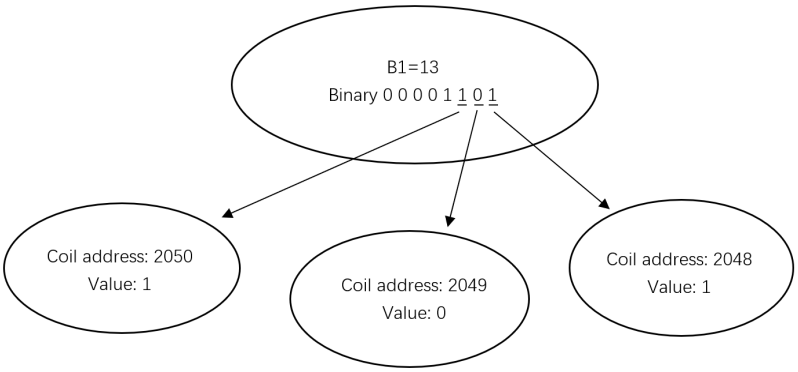
- The values of one or more ModBus slave coils are read, arranged sequentially from low to high, and stored in a specified variable.
- The Byte-type variable Var occupies 8 bits and can accommodate up to 8 coil values.
- The values are stored from the low to the high bits. For example, the coil value of the start address is stored to Bit0 of the variable, the coil value of "start address + 1" is stored to bit1, and so on.
- When there are less than 8 coils, the excessive coil values will not be read, and the values on the corresponding bits are 0.

- **Example**:

Task: Reads value in coil address 2048, 2049.

##Reads two coils from address 2048 and stores them in B1.

Programming: GetModBusCoil(2048,2,B[1]);



11.10 SetModBusCoil

- **Function**: Sets the ModBus slave coil value.
- **Description**: Maps the Byte-type value to the ModBus slave coil value.

- **Format**: SetModBusCoil(StartAddr, CoilNum, ByteValue).
- **Parameter**: StartAddr: Start address of the coil (2048 to 4095) ‖ (6144 to 8191), in bit.

CoilNum: The number of coils to be written (1 to 8)

ByteValue: Byte-type value (0 to 255). Set the coil value of the ModBus slave based on the value of this variable.

> NOTE
>
> - Byte-type value, set from low bit to high bit into one or more coils.
> - The Byte-type value occupies 8 bits and can accommodate up to 8 coil values.
> - The variable is set from low bit to high bit. For example, Bit0 is set to the coil of "start address", Bit1 is set to the coil of "start address+1", and so on.
> - When the number of coils set is less than 8, the excessive bits will not be issued to the coils.
> - The valid range of the coil address is 2048 to 4095 and 6144 to 8191. Coils outside this range results in errors.

- **Example**:

Task: Set the values in coil addresses 2048, 2049, and 2050 to 1, 0, and 1.

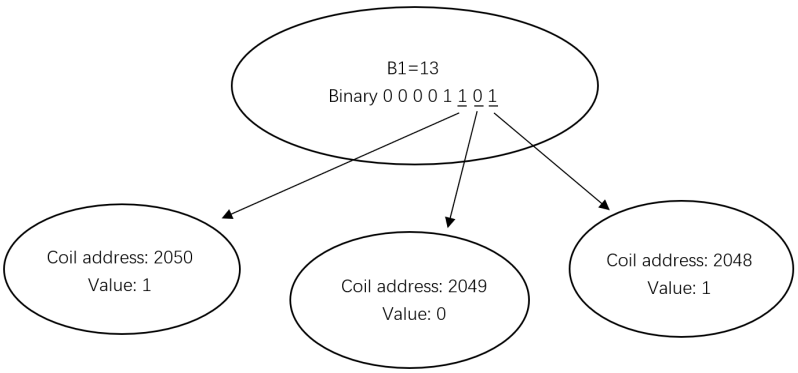
Programming:

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 excessive bits will not be issued.

SetModBusCoil(2048,3, B[1]);



11.11 GetModBusReg

- **Function**: Reads values from register of the ModBus slave.
- **Description**: Reads a number (VarNum) of data from the ModBus register according to the data type specified by DataType, forcibly converts the data to the data type specified by VarName, and stores the data in the variable array specified by VarName.
- **Format**: GetModBusReg(StartAddr,VarName,DataType,VarNum).
- **Parameter**: StartAddr: Start 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/LB/LR/LD arrays only).

DataType: Type of data Read the data according to the data type specified below:

- Short type, which occupies two bytes, that is, one register.
- Int type, which occupies four bytes, that is, two registers.
- Float type, which occupies four bytes, that is, two registers.
- Double type, which occupies eight bytes, that is, four registers.

VarNum: The number of data read. The value ranges from 1 to 256.

Because the data needs to be stored in an array, VarNum + the variable number in VarName ≤ the length of the array.

- **Example**:

GetModBusReg(16384, R[1],1,2);##Reads two short-type data (2*1 registers) from the start register address 16384, and forcibly converts them to a R variable (int) for storage in R[1] and R[2].

GetModBusReg(16384, R[1],2,3);##Reads three short-type data (3*2 registers) from the start register address 16384, and forcibly converts them to a R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384,R[1],3,3);##Reads three float-type data (3*2 registers) from the start register address 16384, and forcibly converts them to a R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384,R[1],4,3);##Reads three double-type data (3*4 registers) from the start register address 16384 and forcibly convert them to a R variable (int) for storage in R[1], R[2] and R[3].

GetModBusReg(16384,D[1],1,3);##Reads three short-type data (3*1 registers) from the start register address 16384 and forcibly convert them to a D variable (double) for storage in D[1], D[2] and D[3].

11.12 SetModBusReg

- **Function**: Sets the value of the ModBus slave register.
- **Format 1**: SetModBusReg(StartAddr,Value, DataType,VarNum).
- **Description**: Forcibly converts Value to the data corresponding to DataType, and sets a number (VarNum) of data to a Modbus address with a start address StartAddr and a length VarNum*DataType.
- **Format 2**: SetModBusReg(StartAddr,&VarName, DataType,VarNum).
- **Description**: Forcibly converts the data in the variable array specified by VarName to the data corresponding to DataType, and sets a number (VarNum) of data to a Modbus address with a start address StartAddr and a length of VarNum*DataType.
- **Parameter**:

StartAddr: Start address of the register (16384 to 32767, or 49152 to 65535).

VarName: Starting element of the 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: Type of data The data is converted according to the data type specified below:

- Short type, which occupies two bytes, that is, one register.
- Int type, which occupies four bytes, that is, two registers.
- Float type, which occupies four bytes, that is, two registers.
- Double type, which occupies eight bytes, that is, four registers.

VarNum: The number of data written. The value ranges from 1 to 256.

Because the data needs to be stored in an array, VarNum + the variable number in VarName ≤ the length of the array.

Example: 20 + 100 ≤ 255 in SetModBusReg(16384, &R[100],1,20)

- **Example**:

SetModBusReg(16384, &R[1],1,2);##Forcibly converts the values of R[1] and R[2] into short-type data and stores them to register addresses from 16384 to 16384 + 2*1-1.

SetModBusReg(16384, &R[1],2,3);##Forcibly converts the values of R[1], R[2], and R[3] into int-type data and stores them to register addresses from 16384 to 16384 + 3*2-1.

SetModBusReg(16384,&R[1],3,3);##Forcibly converts the values of R[1], R[2], and R[3] into float-type data and stores them to register addresses from 16384 to 16384 + 3*2-1.

SetModBusReg(16384,&R[1],4,3);##Forcibly converts the values of R[1], R[2], and R[3] into double-type data and stores them to register addresses from 16384 to 16384 + 3*4-1.

SetModBusReg(16384,20,4,3);##Forcibly converts 20 into double-type data (20.000) and stores it to register addresses from 16384 to 16384 + 3*4-1. All three data stored in this address are 20.000.

SetModBusReg(16384,R[1],4,3) ##Forcibly converts the value of R[1] into double-type data (20.000) and stores it to register addresses from 16384 to 16384 + 3*4-1. All three data stored in this address are values of R[1].

11.13 UIDialog

- **Function**: Messagebox instruction.
- **Description**: Pops up dialog box for user to make selections and identifies the user selections.
- **Format**: Int iRet = UIDialog(caption, content, btntype).
- **Parameter**:

caption: Pop-up box title, type: string. Up to 30 characters are supported.

content: Body content, type: string. Up to 50 characters are supported.

btntype: Button style, type: int. Currently, six types of buttons are supported: 0 for OK button; 1 for Abort, Retry, and Ignore buttons; 2 for OK and Cancel buttons; 3 for Retry and Cancel buttons; 4 for Yes and No buttons; and 5 for Yes, No and Cancel buttons.

- **Return value**:

The button type selected, int.

Normal return value: 1 for OK; 2 for Abort; 3 for Retry; 4 for Ignore, 5 for Cancel, 6 for Yes, and 7 for No.

Abnormal return value: -1 for format error; -2 for network disconnection; -3 for task stop; -4 for command parameter out of range.

Due to the configuration pop-up box, the button style 1 indicates the Abort, Retry, and Ignore buttons for the teach pendant, and the Abort, Retry, and Ignore buttons for the InoRobotLab; and the return value 2 indicates the Abort button for the teach pendant, and the Abort button for the teach pendant and the InoRobotLab. The two are only text differences, and the method of use is no different.

- **Example**:

Int iRet = UIDialog ("Title","Content",2); #Displays a pop-up box including OK and Cancel buttons

This instruction cannot be used in multitasking.

12 Gravity Load-Related Instructions

12.1 GripLoad

- **Function**: Sets the current workobject load number.
- **Description**: Sets the current workobject load number. After the instruction is executed, the active workobject load number is synchronized to the controller.
- **Format**: GripLoad ObjNo.
- **Parameter**: ObjNo: Active workobject load number, integer value, 0 to 15.

GripLoad 0;

This instruction activates workobject 0. The workobject 0 cannot be edited. Therefore, activating workobject 0 is equivalent to remove the workobject load.

- **Example**:

GripLoad 6

If the load number to be switched is not Load 0 and the load mass is 0, the switchover fails.

13 Current Protection-Related Instructions

13.1 AvgCurLmt

- **Function**: Configures average load rate limit.
- **Description**: Sets whether to enable the average load rate detection and sets the average load limit parameters in the program. This instruction is applicable to a single axis, or to all axes.
- **Format**: AvgCurLmt (Enable,AxisNo,ratio).
- **Parameter**: Enable: Specifies whether the instruction is active or not by integer data. 1: Active; 0: Inactive.

AxisNo: Specifies the axis number with integer data. The value range is 0 to 6. Special case: You can use "0" to make all axes the object.

ratio: Specifies the local limit coefficient of the single-axis average load rate with integer data. The unit is 1% and the value range is 1 to 150. If Enable is "0", this parameter is invalid.

Properties of the detection switch:

1. The detection switch on the setting interface of the teach pendant defaults to "Open", and the default setting 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. For the situations where the default setting is restored, see 2.4 Scope of Variables and Instructions.

This command only takes effect on the robot manipulator axes. For example: When using a four-axis controller and connecting external axes, setting the AxisNo parameter to 5 or 6 will cause this instruction to have no effect.

13.2 MaxTrqLmt

- **Function**: Configures the maximum torque limit (current limit).
- **Description**: Sets whether to enable the current detection and sets the current limit parameters in the program. This instruction is applicable to a single axis, or to all axes.
- **Format**: MaxTrqLmt (Enable,AxisNo,ratio).
- **Parameter**: Enable: Specifies whether the instruction is active or not by integer data. 1: Active; 0: Inactive.

AxisNo: Specifies the axis number with integer data. The value range is 0 to 6. Special case: You can use "0" to make all axes as the object.

ratio: Specifies the local limit coefficient of the single-axis current with integer data. The unit is 1% and the value range is 1 to 150. If Enable is "0", this parameter is invalid.

Properties of the detection switch:

1. The detection switch on the setting interface of the teach pendant defaults to "Open", and the default setting 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. For the situations where the default setting is restored, see 2.4 Scope of Variables and Instructions.

This command only takes effect on the robot manipulator axes. For example: When using a four-axis controller and connecting external axes, setting the AxisNo parameter to 5 or 6 will cause this instruction to have no effect.

14 Collision Detection-Related Instructions

14.1 SetCollMode

- **Function**: Sets the status of collision detection switch in auto mode and the actions triggered after collision alarm occurs.
- **Description**: Sets the status of collision detection switch in auto mode and the actions triggered after collision alarm occurs. The parameters set by this instruction take effect together with the parameters set in the user interface (The alarm triggered actions are solely subject to the parameters set by the instruction.)
- **Format**: SetCollMode(value,iMode).
- **Parameter**:

value:

ON to enable the collision detection, OFF to disable the collision detection.

iMode:

Action triggered after collision alarm occurs, including Coll_Default, Coll_PlanStop, Coll_Rebound, and Coll_RapidStop, as explained below:

  - Coll_Default: The action is subject to the user interface setting.
  - Coll_PlanStop: The robot stops as planned.
  - Coll_Rebound: The robot moves back.
  - Coll_RapidStop: The robot immediately stops.

Numbers/B/R/LB/LR variables are supported. The value of these variables can only be 0, 3, 4, and 7. The value 0 corresponds to Coll_Default, 3 to Coll_PlanStop, 4 to Coll_RapidStop, and 7 to Coll_Rebound.

- This instruction takes effect only in the auto mode.
- For initialization conditions, see 2.4 Scope of Variables and Instructions.

- **Example**:

SetCollMode (ON,3);

SetCollMode (ON,Coll_Rebound);

14.2 SetAxisCollMode

- **Function**: Sets the single-axis collision detection switch in the auto mode.
- **Description**: Sets the collision detection switch for the single axis in the auto 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).
- **Parameter**: iAxisNo: Axis number, integer data, range 1 to N, N being the number of robot axes.

Value: Collision detection switch flag, ON or OFF.

CAUTION

- This instruction takes effect only in the auto mode.
- For initialization conditions, see 2.4 Scope of Variables and Instructions.

- **Example**:

SetAxisCollMode(2,OFF);

14.3 SetAxisCollLevel

- **Function**: Sets the single-axis collision detection switch in auto mode.
- **Description**: Sets the collision detection sensitivity of the single axis in the auto 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 . 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).
- **Parameter**: iAxisNo: Axis number, integer data, range 1 to N, N being the number of robot axes.

iPara_Level: Axis collision detection sensitivity in auto mode, real number, 25 to 300

CAUTION

- This instruction takes effect only in the auto mode.
- For initialization conditions, see 2.4 Scope of Variables and Instructions.

- **Example**:

SetAxisCollLevel (1, 157.258);

14.4 SetCollDist

- **Function**: Sets the move back distance (mm) after collision detection in program mode.
- **Format**: SetCollDist(Dist).
- **Parameter**: Dist: Distance, unit mm, integer data.

CAUTION

- This instruction takes effect only in the auto mode.
- For initialization conditions, see 2.4 Scope of Variables and Instructions.
- This parameter only takes effect in "Immediate rollback" mode.
- SetCollDist does not take effect when called without the trigger action set to rollback.
- If the running distance before collision is less than the set value, it will move back to the initial position.

- **Example**:

SetCollMode (ON, Rebound); # Set collision action to immediate moving back.

SetCollDist (10); Set 10 mm move back distance after collision.

15 Conveyor-Related Instructions

15.1 CnvVision

- **Function**: Opens/Closes the vision port of the conveyor.
- **Description**: Opens/closes the vision port of the specified conveyor. With the vision port open, the controller automatically stores the objects detected by the vision unit into a queue, without the need for the user to program the vision coordinates.
- **Format**: CnvVision(Conveyor[Index],ON|OFF,ClientPort).
- **Parameter**: Index: Conveyor index (0 to 3).

ON|OFF: Opens or closes the vision port.

ClientPort: Client port number, 2 to 5, or 1024 to 65535.

2 to 5 indicate the first to the 4th socket connections. In this case, you do not need to specify the client port number.

1024 to 65535 indicate the socket connections with specified client port number.

- **Example**:

CnvVision(Conveyor[0],ON,1024); #Opens the vision port of conveyor 0, client port number 1234.

L[0]:

GetCnvObject(0,0),Goto L[0];

……

CnvVision(Conveyor[0],OFF,1024); #Closes the vision port of conveyor 0.

- The CnvVision instruction must be used after the network communication establishment and before GetCnvObject instruction.
- Once you have opened a vision port using the CnvVision instruction, the Get Port, Send Port, GetAString or SetSocketRecvType instruction can no longer be used on this port.
- The vision unit can be triggered to take pictures only after the port is opened using this instruction. Therefore, if you switch running mode of the robot when running the dynamic tracking program, the port will be closed. When switching back to the auto mode, you need to open the port again simply by returning to the start line and running this instruction again.
- When the one-drive-many vision function is enabled and the current robot controller acts as the master, all slave controllers must be connected to the master successfully before the instruction can be executed.
- When the one-drive-many vision function is enabled and the current robot controller acts as the slaver, CnvVision(*, ON, *) is not allowed.

15.2 Cnv

- **Function**: Sets the keywords of the Wobj frame for the tracking process.

- **Description**: It is an additional parameter for the Movl, Movc, Movs, Jump, and JumpL instructions for the tracking process.
- **Format**:

RefSys Conveyor(1,Tool[2]);

1. Movl P[30],V[100],Z[1],Tool[2],**Cnv**;
2. Movc P[30],V[100],Z[1],Tool[2],**Cnv**;
3. Movs P[0:30],V[100],Z[1],Tool[2],**Cnv**;
4. umpL P[30],V[100],Z[1],Tool[2],**Cnv**;

RefSys Base;

- **Range**: It can only be used in motion instructions of the tracking process.

15.3 GetCnvObject

- **Function**: Queries whether there are objects of the specified type in the pickup area of the designated conveyor.
- **Description**: Queries the objects queue for an object of the specified type in the pickup area (between the upper workspace boundary and the lower pickup boundary). If the object exists, the object is taken from the queue and then deleted from the queue, if not, the program jumps.
- **Format**: GetCnvObject(CnvID,ObjID),Goto L[Index].
- **Parameter**: CnvID: Conveyor index (0 to 3).

ObjID: Object type index (0 to 15).

Index: The index of label to which the program jumps if no query result is received within the specified time.

1. When the detection mode is "vision", the GetCnvObject instruction is used together with the CnvVision instruction. After the vision port is opened, the received vision data is sent directly to the DSP for processing until the vision port is closed. When the detection mode is "sensor", the GetCnvObject instruction is used separately.
2. If an object is found, the program proceeds to the next line of instructions (the Goto L[Index] instruction will not be executed). If no object is found, the Goto L[Index] instruction will be executed. That is, the program jumps to L[Index].
3. Calling GetCnvObject in the loop allows you to continuously obtain the position data of objects that enter the pickup space on the conveyor. Each time GetCnvObject is used, it will retrieve a piece of position data to get the position of the object on the conveyor that matches the object type. The next time it is used, it will automatically retrieve the next piece of position data to get the position of the next object on the conveyor that matches the object type.
4. When the detection mode is "vision", the conveyor index in GetCnvObject/CopyCnvObject/ ClearCnvObject instructions must be the same as that in the CnvVision instruction.
5. The GetCnvObject and CopyCnvObject instructions are not allowed for multi-tasking because the two instructions will obtain the object information at high speed, resulting in high CPU usage.

CAUTION

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 GetCnvObject instruction fetches them. Therefore, when the current motion is not complete, the next position data can still be retrieved. Therefore, the next object will not be lost due to the motion delay, unless the position of the object on the conveyor is out of boundaries.

- **Example**:

```
Start;
##Open the port. The external device acts as the server, with an IP 10.45.151.108 and a port 2000.
##The local controller acts as a client and the port number is 1234.
R[0] =1234;
B[0] =0;
While B[0] <> 1
Open Socket("10.45.151.108",2000,R[0],Single,B[0]);
EndWhile;
CnvVision(Conveyor[1],OFF,R[0]);
CnvVision(Conveyor[1],ON,R[0]);                    ##Opens the vision port of conveyor 1, client port number 1234
P[30]=(0,0,10,0,0,0),(0,0,0,0),(0,0,0,0,0,0);      ##Defines P[30] as a point in the object frame, with coordinates (0,0,10,0,0,0)
L[0]:
Movj P[0],V[100],Z[0],Tool[0],Wobj[0];             ##Moves to the wait position
L[1]:
GetCnvObject(1,0), Goto L[1];                      ##Queries the queue for object of type 0 on conveyor #1
RefSys Conveyor(1,Tool[2]);                        ##Switches the motion reference frame to the object that has just been found
Movl P[30],V[100],Z[1],Tool[2],Cnv;                ##Moves to the P[30] in the object frame by using tool 2
Delay T[1];                                        ##Keeps the robot moving with the conveyor for 1s
Set Out[1],ON;                                     ##Turns on the switch to suck the object
RefSys Base;                                       ##Switches the motion reference system to the robot
Jump P[1],V[100],Z[0],Tool[0],Wobj[0],LH[10],MH[-750],RH[10];    ##Moves the object to P[1]
Set Out[1],OFF;                                    ##Places the object
Goto L[0];
End;
```

15.4 CopyCnvObject

- **Function**: Queries whether there are objects of the specified type in the pickup area of the designated conveyor.

- **Description**: Queries the objects queue for an object of the specified type in the pickup area (between the upper workspace boundary and the lower pickup boundary). If the object exists, the object is copied and still kept in the queue, if not, the program jumps.
- **Format**: CopyCnvObject(CnvID, ObjID),Goto L[Index].
- **Parameter**: CnvID: Conveyor index (0 to 3).

ObjID: Object type index (0 to 15).

Index: The index of label to which the program jumps if no query result is received within the specified time.

NOTE

- This instruction is the same with the GetCnvObject format and syntax. The difference is that CopyCnvObject copies the object from the queue and the object still remains in the queue, while GetCnvObject takes the object from the queue and the deletes the object from the queue.
- This instruction is suitable for the scenario where it is needed to query the same object several times. For example, you need to perform dynamic tracking operations on the same object twice. When the object is successfully queried for the first time, the tracking operation A is performed. Then the tracking operation B is performed when the object is successfully queried for the second time. Then the tracking process is ended (RefSys Base).

- **Example**:

L[1]:

CopyCnvObject(1,0),Goto L[1]

15.5 RefSys

- **Function**: Switches the motion reference frame.
- **Description**: When external motion mechanisms are used, such as conveyors or workbenches, it is necessary to identify which frame the position variables are in. In this case, use RefSys to switch to the external frame. When it is necessary to use the position variables in the robot frame, use Refsys again to switch back to the robot frame.
- **Format 1**: RefSys Base.

NOTE

- Switch to the base frame (static frame). When you switch back from the conveyor or workbench frame to the robot frame, the robot motion stops.
- After the RefSys Base instruction is executed, the current tool number of the system changes to 0.

- **Format 2**: RefSys Conveyor(CnvID,Tool[*]).
- **Parameter**:

CnvID: Conveyor index (0 to 3).

Tool[*]: Tool index, indicating that the conveyor motion is synchronized with the specified TCP.

NOTE

- Switch to the object frame on the conveyor. Because the object frame is dynamic, the TCP will follow the conveyor motion after this instruction is executed.
- After the RefSys Conveyor instruction is executed, the current tool number of the system changes to the tool number written in the Tool parameter.
- After the RefSys Conveyor instruction is executed, the motion of the robot is relative to the object frame. When no motion instruction is applied, the robot and the object remain relatively static (i.e., the robot moves in synchronization with the object). When a motion instruction is applied, the motion trajectory is a trajectory under the object frame, and the target point is a coordinate point under the object frame.

CAUTION

- The PE parameter is invalid for the conveyor tracking process. In other words, the PE parameter must not be included in motion instructions during the conveyor tracking process.
- The workobject parameters of the motion instructions between the RefSys Conveyor and RefSys Base instructions use the keyword Cnv, not Wobj. The motion instructions include Movl, Movc and JumpL.
- The RefSys Conveyor and RefSys Base instructions must be used together.

- **Example**:

```
Start;
……
P[30]=(0,0,10,0,0,0),(0,0,0,0),(0,0,0,0,0,0);      ##Defines P[30] as a point in the object frame, with coordinates (0,0,10,0,0,0)
Movj P[0],V[100],Z[0],Tool[0],Wobj[0];             ##Moves to the wait position
L[2]:
GetCnvObject(1,0), Goto L[2];                      ##Queries the queue for object of type 0 on conveyor #1
RefSys Conveyor(1,Tool[2]);                        ##Switches the motion reference frame to the object that has just been found
Movl P[30],V[100],Z[1],Tool[2],Cnv;                ##Moves to the P[30] in the object frame
RefSys Base;                                       ##Switches the motion reference system to the robot
……
End;
```

15.6 ClearCnvObject

- **Function**: Clears object from conveyor.
- **Description**: Clears one or all objects on the specified conveyor. Detected objects on the conveyor are not automatically cleared when the program is paused or stopped, and need to be cleared using this instruction.
- **Format**: ClearCnvObject(CnvID,Num|ALL).

- **Parameter**: CnvID: Conveyor index (0 to 3).

Num|ALL: The number of objects to be cleared.

> NOTE
>
> The conveyor index in GetCnvObject/CopyCnvObject/ClearCnvObject instructions must be the same as that in the CnvVision instruction.

- **Example**:

##Clears two object data from conveyor 1

ClearCnvObject(1,2);

##Clears all object data from conveyor 1

ClearCnvObject(1,ALL);

15.7 CnvFastCatch

- **Function**: Realizes tracking in fast pickup operation.
- **Description**: Moves through Jump motion to a point in the frame of the object on the conveyor while keeping synchronization with the conveyor motion.
- **Format**: CnvFastCatch P, V, Z, CnvF(CnvID, ON/OFF), Tool,[Acc], [Dec], [NWait], [Out(No,value,Type)…], LH, MH, RH.
- **Parameter**: This instruction is consistent with the CnvF instruction in parameters except CnvF. For details, see the Jump instruction.

Mandatory parameters:

P: Target point to be reached, point number range 0 to 9999, can be a global point or local point (P|LP).

V: Specified speed, percentage value, range 1 to 100.

Z: Arrival and transition parameter. Options include Fine, Z[0] to Z[200], Z[CP], and ZR[m].

CnvF (CnvID, ON/OFF): Fast pickup mode parameter setting (mandatory parameters)

CnvID: Conveyor index (0 to 3)

ON/OFF: Time-first mode. When ON, the robot will select the optimal pickup path to achieve the shortest pickup time. OFF option is currently not available.

Tool: Tool frame, range Tool[0] to Tool[15], required.

Optional parameters:

Acc: Specifies the acceleration. Percentage value, 20 to 100

Dec: Specifies the deceleration. Percentage value, 20 to 100

Nwait: No-wait flag.

Out(No,value,Type)…: Outputs signals in parallel in running.

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: MH[Z]. The value range is -2000.000 to 2000.000. The Z value represents the coordinate value Z in the robot base frame.

CAUTION

- This instruction does not support the self-learning vibration control function.
- This instruction is only applicable to the SCARA robots and Delta robots.
- The workobject parameters of the motion instructions between the CnvFastCatch and RefSys Base instructions use the keyword Cnv, not Wobj. The motion instructions include Movl, Movc and JumpL.

- **Example**:

```
Start;
##Open the port. The external device acts as the server, with an IP 10.45.151.108 and a port 2000.
##The local controller acts as a client and the port number is 1234.
R[0] =1234;
B[0] =0;
While B[0] <> 1
Open Socket("10.45.151.108",2000,R[0],Single,B[0]);
EndWhile;
CnvVision(Conveyor[1],OFF,R[0]);
CnvVision(Conveyor[1],ON,R[0]);                ##Opens the vision port of conveyor 1, client port number 1234
P[30]=(0,0,10,0,0,0),(0,0,0,0),(0,0,0,0,0,0);    ##Defines P[30] as a point in the object frame, with coordinates (0,0,10,0,0,0)
P[31]=(0,0,50,0,0,0),(0,0,0,0),(0,0,0,0,0,0);    ##Defines P[31] as a point in the object frame, with coordinates (0,0,50,0,0,0)
L[0]:
Movj P[0],V[100],Z[0],Tool[0],Wobj[0];          ##Moves to the wait position
L[1]:
GetCnvObject(1,0), Goto L[1];                   ##Queries the queue for object of type 0 on conveyor #1
CnvFastCatch P[30], V[30], Z[0], CnvF(1 OFF), Tool[2], LH[10], MH[-10] RH[20];      ##Moves to a point P[30] in the object frame by using tool 2 and switches the motion reference frame to the object on the conveyor just found (i.e., keeping synchronization with the conveyor)
Delay T[1];                                      ##Keeps the robot moving with the conveyor for 1s
Set Out[1],ON;                                  ##Turns on the switch to suck the object
Movl P[31],V[100],Z[1],Tool[2],Cnv;              #Moves to the P[31] in the object coordinate system by using tool 2, which is a lifting action
RefSys Base;                                    ##Switches the motion reference system to the robot
Jump P[1],V[100],Z[0],Tool[0],Wobj[0],LH[10],MH[-750],RH[10];  ##Moves the object to P[1]
Set Out[1],OFF;                                 ##Places the object
Goto L[0];
End;
```

15.8 GetCnvObjType

- **Function**: Returns the type index of the first object in the queue to be picked.
- **Format**: Int ObjID= GetCnvObjType(CnvID).
- **Parameter**: CnvID: Conveyor index (0 to 3).

ObjID: Object type index, Int type (-1 to +15).

Return value -1 indicates that the queue contains no object.

Return value 0 to 15 indicates the type of the object most downstream in the queue.

- **Example**:

##Get the type index of the first object in the queue of conveyor 1.

Int objID;

objID = GetCnvObjType(1);

15.9 GetCnvSpeed

- **Function**: Returns the current speed of the specified conveyor, in mm/s or degree.
- **Format**: Double cnvSpeed = GetCnvSpeed(CnvID).
- **Parameter**: CnvID: Conveyor index (0 to 3).

cnvSpeed: Current conveyor speed, double type

- **Example**:

##Get the current speed of conveyor 1

Double cnvSpeed;

cnvSpeed= GetCnvSpeed(1);

15.10 GetCnvCntPos

- **Function**: Returns the current position of the conveyor encoder, in pulse.
- **Format**: Double EncPos = GetCnvCntPos (CnvID).
- **Parameter**: CnvID: Conveyor index (0 to 3).

EncPos: Current position of the conveyor encoder, double type

- **Example**:

##Get the current position of the encoder of the conveyor 1

Double encPos;

encPos = GetCnvCntPos (1);

15.11 GetCnvObjOutNum

- **Function**: Gets the number of out-of-space objects of the specified workobject type (the number of objects that go beyond the lower pickup boundary and have not been picked up).
- **Format**: Int OutNum = GetCnvObjOutNum(CnvID, objType|ALL).
- **Parameter**: CnvID: Conveyor index (0 to 3).

ObjType|ALL: Object type to be queried.

ObjType: Specifies the object type, rang 0 to 15.

ALL: All object types.

OutNum: Number of out-of-space objects, Int type.

> NOTE
>
> After reading the number of out-of-space objects of the specified workobject type, the number is automatically cleared and the counting starts again.

- **Example**:

##Get the number of out-of-space objects of type 0 on the conveyor 1

Int outNum;

outNum = GetCnvObjOutNum(1,0);

##Get the number of all out-of-space objects on the conveyor 1

Int outNum;

outNum = GetCnvObjOutNum(1,ALL);

15.12 CnvTrigVis

- **Function**: Triggers I/O to trigger camera to take a photo.
- **Format**: CnvTrigVis(Index).
- **Parameter**: Index: Conveyor index (0 to 3).
- **Example**:

##Trigger photo-taking of conveyor 1

CnvTrigVis(1);

> CAUTION
>
> When this instruction is executed, it will trigger a change in the DO level corresponding to the specified conveyor, thereby triggering the camera to take a photo. The triggering period depends on the on-site working conditions, at least 0.6s, to ensure sufficient time for vision data processing and transmission.

15.13 GetCnvObjData

- **Function**: Gets information about the object with the specified serial number in the queue.
- **Description**: Information about the object includes the following:

(1) Object type.

(2) Position of the conveyor encoder (at the moment of sensing).

(3) Distance from the upper workspace boundary (at the moment of sensing or at the current moment).

(4) The coordinates of the object in the conveyor frame (at the moment of sensing or at the current moment).

- **Format**: TCnvObjData Tt; ##Custom struct type TCnvObjData.

GetCnvObjData (CnvID,Num,Origin|Current,Tt);

- **Parameter**: CnvID: Conveyor index (0 to 3).

Num: Serial number of the specified object in the queue, rang 0 to 500.

0: Object being picked up

1-500: Objects to be picked up in the queue

Origin | Current: Information about object at the moment of sensing or at the current moment

Origin: Moment of sensing (camera's photo taking moment or sensor's sensing moment)

Current: Current moment

Tt: Return value, information about the object, struct type TCnvObjData.

Struct TCnvObjData

{

Int objType; ##Object type

Double encVal; ##Instantaneous encoder value at the moment of sensing

Double distance; ##Distance from the upper workspace boundary (at the moment of sensing or at the current moment)

Double posOnCnv[6]; ##Coordinates of the object in the conveyor frame (at the moment of sensing or at the current moment)

}

EndStruct;

- **Example**:

##Define the object information struct, strictly following the variable types and order below, and the variable names can be freely chosen.

Struct TCnvObjData

Int objType; ##Object type

Double encVal; ##Instantaneous encoder value at the moment of sensing

double lenth; ##Distance from the upper workspace boundary (at the moment of sensing or at the current moment)

Double posOnCnv[6]; ##Coordinates of the object in the conveyor frame (at the moment of sensing or at the current moment)

EndStruct;

Start;

TCnvObjData Tt; ##Defines object information struct variables

R[0] =1234;

B[0] =0;

While B[0] <> 1

Open Socket("10.45.151.108",2000,R[0],Single,B[0]);

EndWhile;

```
CnvVision(Conveyor[1],OFF,R[0]);

CnvVision(Conveyor[1],ON,R[0]);

P[30] = (0,0,10,0,0,0),(1,0,0,0),(0,0,0,0,0,0); ##Defines P[30] as a point in the object frame, with coordinates (0,0,10,0,0,0)

L[0]:

Movj P[0],V[100],Z[1],Tool[0];      ##Moves to the wait position

GetCnvObjData (0, 1, Current, Tt); ##Gets the current moment information of the first object on the queue of conveyor 0 into the variable Tt

Print("Obj:"+Tt.objType);           ##Prints the object type

Print("encVal:"+Tt.encVal); ##Prints the encoder value of the object at the time of photo taking

Print("lenth:"+Tt.lenth);           ##Prints the distance of the object from the upper workspace boundary at the current time

Print("posOnCnv[0]:"+Tt.posOnCnv[0]); ## Prints the coordinate X of the object in the conveyor frame at the current time

L[1]:

GetCnvObject(0,8),Goto L[1];

RefSys Conveyor(0,Tool[0]);

Movl P[30],V[30],Z[0],Tool[0],Cnv;

RefSys Base;

Goto L[0];

End;
```

- When the object type objType in the return value is -1, it indicates that the object information acquisition fails or the object is empty.
- When Num = 0, it indicates the information of the current picked up object information. However, because the GetCnvObject instruction updates the current picked up object, the following two conditions exist:
  - Incorrect use: When the GetCnvObjData instruction is before the GetCnvObject instruction or after the RefSys Base instruction, the information obtained is empty, indicating that the current object has already been picked up.
  - Correct use: When the GetCnvObjData instruction is between the GetCnvObject instruction and the RefSys Base instruction, the information obtained is the information of the object being picked up.
- The struct of the returned object information must be defined as (1 int + 8 double); otherwise, an error occurs.
- Coordinate value X of the object - Upper workspace boundary = Distance of the object from the upper workspace boundary. Therefore, when the distance is a negative value, the object has not entered the upper workspace boundary. When the distance is a positive value, the object has entered the upper workspace boundary.

15.14 SetCnvVisOTMode

- **Function**: Sets the vision timeout mode in the tracking process.
- **Description**: When the tracking process is used, if the robot triggers the nth photo taking, the robot must receive the vision data of the nth photo taking before the nth+1st photo taking is triggered, in order to ensure the real-time vision data. Otherwise, an error 0x2025 occurs.

For certain applications with low requirements or poor working conditions, it is necessary to allow the vision data to not be sent in time. In this case, you can set the vision timeout mode through this instruction. You can define the number of consecutive times that vision data is not sent in time.

- **Format**: SetCnvVisOTMode (CnvIndex,AlarmMode,Num).
- **Parameter**: CnvIndex: Conveyor index (0 to 3)

AlarmMode: Defines the form of alarm upon detection that the vision timeout threshold has been reached.

Warn: Warning, the robot does not stop.

Alarm: Alarm, the robot stops.

Num: If the vision data is not received for consecutive Num (≥ 1) times, a warning or an alarm occurs.

Example:

#Trigger alarm 0x2025 when the vision data of conveyor 1 is not sent in time for three consecutive times.

SetCnvVisOTMode(1,Alarm,3).

#Trigger warning 0x50D2 when the vision data of conveyor 2 is not sent in time for two consecutive times.

SetCnvVisOTMode(2,Warn,2).

CAUTION

- If the robot has not executed this instruction before, the default values inside the robot are AlarmMode = Alarm and Num = 1. That is, an alarm occurs if the vision data is not sent in time for once.
- This instruction only needs to be executed for one time to take effect, so it only needs to be used at the beginning of the program.
- After this instruction is executed, the parameters set remain in effect until the following two situations occur:
  - The controller is restarted. In this case, the parameters set are restored to defaults.
  - Parameters are updated using this instruction again.

15.15 GetCnvVisOTNum

- **Function**: Gets the number of times the vision data transmission times out in the tracking process.
- **Description**: After using the instruction SetCnvVisOTMode to set the condition that an alarm or warning is triggered only after n (n>1) consecutive failures to receive vision data, the robot will start counting the total number of times that the vision data transmission is delayed.
- **Format**: R[*] = GetCnvVisOTNum (CnvIndex).
- **Parameter**: CnvIndex: Conveyor index (0 to 3)

R[*]: Return value: The total number of times the vision data transmission times out.

- **Example**:

#Get the number of times the vision data of conveyor 1 is not sent in time.

R[0] = GetCnvVisOTNum(1);

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 flying trigger function. After position latch is turned on, the user I/O Out[14] or Out[15] is used to trigger photo taking 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.

- The IRCB501, IRCB500 and IRCB300 series controllers support the position latching function. You only need to connect the corresponding I/O to the camera as a trigger for taking photos.
- The IRCB10 series controller does not support the flying trigger function.
- The IRCB300 series controller position latching triggered by one signal (fixed to "Out[15]"). To prevent the position latching function from being unavailable due to damage, the IRCB500 and IRCB501 series controller support position latching triggered by two signals: Out[14] or [Out15], and the default is Out[14].

16.1 LatchEnable

- **Function**: Turns on/off position latching function.
- **Description**: When position latching 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 the GetLatchPos instruction.
- **Format**: LatchEnable ON|OFF.
- **Parameter**: ON|OFF: ON for enable, and OFF for disable.
- **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]);
EndWhile;
Print P[10];
LatchEnable OFF;
End;
```

CAUTION

After using the position latch function, you need to turn it off by LatchEnable OFF.

16.2 ClearLatchPos

- **Function**: Clears latched positions.
- **Description**: Clears previously latched robot positions.
- **Format**: ClearLatchPos.
- **Parameter**: /.
- **Example**: See LatchEnable.

16.3 GetLatchPos

- **Function**: Reads latched positions.
- **Description**: Reads last latched robot position. The position points of the tool, including the external axis values are read.
- **Format**: GetLatchPos(P/LP).
- **Parameter**: P/LP: Position variable that stores the obtained results.
- **Return value**: Returns 0 for no latched position data and 1 latched position data. The value of P/LP is valid when 1 is returned.

NOTE

- The robot controller supports caching up to 10 latched position 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.
- When multiple latched position data is triggered, the GetLatchPos instruction 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]);
EndWhile;
Print P[10];
```

```
B[2]=0;
While B[2]==0
    B[2] =GetLatchPos(P[11]);
EndWhile;
Print P[11];
LatchEnable OFF;
End;
```

For initialization conditions, see 2.4 Scope of Variables and Instructions.

17 Temperature Rise Protection-Related Instructions

17.1 GetEncoderTemper

- **Function**: Gets the current temperature information of the encoder.
- **Description**: This instruction is used to obtain the current encoder temperature.
- **Format**: Double temper = GetEncoderTemper (index).
- **Parameter**: index: Specifies an axis of the robot, range 1 to 8
- **Return value**: double type, the temperature of the ith axis encoder. An alarm occurs when the encoder temperature is wrong.
- **Example**:

D[0] = GetEncoderTemper (1).

- This instruction can only be used in robot models that support the temperature rise protection function.
- In the following situations, the encoder temperature is valid: The servo parameter H0B66 (encoder temperature) is not 0 and H0004 (encoder version) is not 2312.2.
- When the robot does not support the temperature rise protection function, an error code -505 is returned. If the input axis number is incorrect (not between 1 and 8), an error code -404 is returned.
- For SCARA models, inputting axis numbers 5 to 8 returns a value of 0; for six-axis models, inputting axis numbers 7 to 8 returns a value of 0. If the return value type uses an integer variable (B/LB/R/LR), the return value type will be forcibly converted to an integer.

18 Independent Axis-Related Instructions

18.1 IndCMove

- **Function**: Enables continuous motion of the independent axis.
- **Description**: The functions of this instruction are as follows:

1. Switches the axis specified in the instruction to an independent axis.
2. Switches the motion mode of the axis to speed mode.
3. Sets the continuous motion speed of the axis and starts continuous rotational motion of the axis.

- **Format**: IndCMove MecUnit, Axis, Speed, [Acc], [Dec].
- **Parameter**: MecUnit: Name of the mechanical unit. If the mechanical unit is a robot, MecUnit is "Robot".

Axis: Axis number. For six-axis robots, only 6 is supported. For other mechanical units, it depends on the actual conditions. It does not support variable input.

Speed: Numerical, positive and negative signals the direction. The specified speed is affected by the global speed percentage. The formula is: Running speed = Speed set by the instruction x Global speed percentage, in °/s, or mm/s for linear axes.

Speed = 0 indicates that continuous motion of the independent axis stops.

Value range: -10000°/s to 10000°/s, default 100

When Speed exceeds the upper limit of the controller speed, the upper limit of the controller speed is prevails.

Speed supports variables.

Acc: (Optional parameter) Specifies acceleration. For example, Acc = 50 represents 50% of the maximum acceleration of the independent axis.

When not specified, the default acceleration is Acc = 50. The minimum effective value is 20%.

- **Range**: 20 to 100.

Dec: (Optional parameter) Specifies deceleration. For example, Dec = 50 represents 50% of the maximum deceleration of the independent axis.

When not specified, the default deceleration is Dec = 50. The minimum effective value is 20%.

Range: 20 to 100.

- **Example**:

START;

IndCMove Robot, 6, 50; ##The J6 axis of the robot rotates continuously at a speed of 50°/s.

IndCMove NoModels, 1, 50; ##The axis 1 of the mechanical unit NoModels rotates continuously at a speed of 50°/s.

IndCMove Robot, 6, 0; ##The J6 axis of the robot stops continuous rotation.

IndCMove NoModels, 1, 0; ##The axis 1 of the mechanical unit NoModels stops continuous rotation.

End;

18.2 IndSpeed

- **Function**: Checks whether the independent axis reaches the planned speed.
- **Description**: It can be used as an additional instruction in conjunction with Wait.
- **Format**: IndSpeed(string MecUnit, int Axis).
- **Return value**: TRUE for specified speed reached; FALSE for specified speed not reached.
- **Parameter**: MecUnit: Name of the mechanical unit. If the mechanical unit is a robot, MecUnit is "Robot".

Axis: Axis number. For six-axis robots, only 6 is supported. For other mechanical units, it depends on the actual conditions.

- **Range**: It can only be used in the main task.
- **Example**:

START;

IndCMove Robot, 6, 50; ##The J6 axis of the robot rotates continuously at a speed of 50°/s.

Wait IndSpeed(Robot, 6); ##Waits until the speed of the independent axis is reached before executing the next instruction.

IndCMove Robot, 6, 0; ##The J6 axis of the robot stops continuous rotation.

Wait IndSpeed(Robot, 6); ##Waits until the speed of the independent axis is reached before executing the next instruction.

Delay T[1]; ##Used with Delay to ensure that the independent axis speed is fully reached.

End;

18.3 IndReset

- **Function**: Resets the independent axis to normal mode.
- **Format**:

IndReset MecUnit, Axis, Old.

IndReset MecUnit, Axis, RefPos, Direction.

IndReset MecUnit, Axis, RefNum, Direction.

- **Description**:
  - This instruction switches the independent axis to the normal mode. The number of turns of the axis is retained.
  - This instruction switches the independent axis to the normal mode. The number of turns of the axis is adjusted according to RefPos and Direction.
  - This instruction switches the independent axis to the normal mode. The number of turns of the axis is adjusted according to RefNum and Direction.
- **Parameter**:

MecUnit: Name of the mechanical unit If the mechanical unit is a robot, MecUnit is "Robot".

Axis: Axis number. For six-axis robots, only 6 is supported. For other mechanical units, it depends on the actual conditions.

RefPos: RobPos type, used as reference position, only Axis component; supports P, LP, and JP. This position is referenced to the current tool and workobject frames, and must be within the normal operating range.

RefNum: Numeric type, reference position; Unit: °; Range: -720° to +720°. Variable values are supported.

Direction:

Short: Adjusts the multi-turn values of the axis so that the current position is as close as possible to the specified RefPos/RefNum position. (Within ±180°)

Forward: Adjusts the multi-turn values of the axis so that the reference position RefPos/RefNum is on the forward side of the changed current position. (Within +360°)

Backward: Adjust the multi-turn values of the axis so that the reference position RefPos/RefNum is on the backward side of the changed current position. (Within -360°)

- **Range**: It can only be used in the main task.

CAUTION

- When the IndReset instruction is executed, the current independent axis and normal axis must not be in motion.
- After executing the IndCMove instruction at zero speed, pause for 0.2 seconds to ensure the correct state is achieved.
- For the external axis with poor performance, pause for a longer time to ensure that the axis stops completely.

- **Example**:

##The current position was originally 750°.

IndReset NoModels_1Axis,1,300 ,Short; ##Axis 1 in NoModels_1Axis changes back to normal mode, with the logical position set to 390°.

##The current position was originally 750°.

IndReset NoModels_1Axis,1,300,Forward; ##Axis 1 in NoModels_1Axis changes back to normal mode, with the logical position set to 390°.

##The current position was originally 750°.

IndReset NoModels_1Axis,1,300, Backward; ##Axis 1 in NoModels_1Axis changes back to normal mode, with the logical position set to 30°.

19 Soft Float-Related Instructions

19.1 SoftModeAct

- **Function**: Enables the soft float function.
- **Description**: This instruction enables the soft float function and can be directly called in the program. After this instruction is executed, the robot switches from position control mode to soft float control mode. This instruction is a blocking type instruction.
- **Format**: SoftModeAct(SoftNum).
- **Parameter**: SoftNum: The full name of this parameter is Soft Number, indicating that the enabled soft float mode uses the SoftNumth conditional parameter. This parameter needs to be set in advance on the soft float parameter settings page. SoftNum is an integer from 0 to 15.
- **Range**: It can only be used in the main task.
- **Example**:

```
Start;
  Movj P[1],V[15],Z[0],Tool[0];    #Sets the points for which the soft float function is enabled
  SoftModeAct (1);          #Enables the soft float function and calls condition 1 parameters
  Wait T[10];               #The operating time of the soft float function. External forces can be applied.
  SoftFollow;               #Enables the soft float following function
  Wait T[10];               #The operating time of the soft float function. External forces can be applied.
  SoftModeDeact;            #Disables the soft float following function
  Movj P[2],V[15],Z[0],Tool[0];    #Moves the robot to P2
End;
```

19.2 SoftModeDeact

- **Function**: Disables the soft float function.
- **Description**: This instruction disables the soft float function and can be directly called in the program. After this instruction is executed, the robot switches from soft float control mode to position control mode. When the SoftModeAct instruction is enabled, this instruction is a blocking type instruction. This instruction can be used independently. If it is called when the SoftModeAct instruction is not active, the robot functions will not be changed.
- **Format**: SoftModeDeact.
- **Parameter**: No parameter description, called directly in the program.
- **Range**: It can only be used in the main task.
- **Example**:

```
Start;
  Movj P[1],V[15],Z[0],Tool[0];    #Sets the points for which the soft float function is enabled
  SoftModeAct (1);          #Enables the soft float function and calls condition 1 parameters
  Wait T[10];               #The operating time of the soft float function. External forces can be applied.
```

```
  SoftFollow;              #Enables the soft float following function
  Wait T[10];              #The operating time of the soft float function. External forces can be applied.
  SoftModeDeact;           #Disables the soft float following function
  Movj P[2],V[15],Z[0],Tool[0];    #Moves the robot to P2
End;
```

19.3 SoftFollow

- **Function**: Enables the soft float follow mode.
- **Description**: This instruction sets whether to enable the following mode for the soft float function. After the soft float function is enabled, if the robot is pushed and no external force is applied, it will return to the teaching point. However, when this instruction is executed, the current position is regarded as the teaching point, and the robot will not return to the original teaching point even after the external force is removed. When the SoftModeAct instruction is enabled, this instruction is a blocking type instruction. This instruction can be used independently. When it is called before the SoftModeAct instruction is enabled, the robot function is unchanged.
- **Format**: SoftFollow.
- **Parameter**: No parameter description, called directly in the program.
- **Range**: It can only be used in the main task.
- **Example**:

```
Start;
  Movj P[1],V[15],Z[0],Tool[0];    #Sets the points for which the soft float function is enabled
  SoftModeAct (1);          #Enables the soft float function and calls condition 1 parameters
  Wait T[10];              #The operating time of the soft float function. External forces can be applied.
  SoftFollow;              #Enables the soft float following function
  Wait T[10];              #The operating time of the soft float function. External forces can be applied.
  SoftModeDeact;           #Disables the soft float following function
  Movj P[2],V[15],Z[0],Tool[0];    #Moves the robot to P2
End;
```

20 Interference Zone-Related Instructions

20.1 SetInterferZone

- **Function**: Enables or disables the interference zone.
- **Description**: This instruction enables or disables a certain interference zone, and can be called directly in the program. This instruction is a blocking type instruction.
- **Format**: SetInterferZone (id,ON/OFF).
- **Parameter:** id indicates the interference zone number, rang 0 to 15; ON/OFF indicates that the corresponding interference zone is enabled or disabled.
- **Range**: It can only be used in the main task.
- **Example**:

```
Start;
  Movj P[1],V[15],Z[0],Tool[0];
  SetInterferZone (1,ON);          #Enables interference zone 1
  Movj P[2],V[15],Z[0],Tool[0];
  SetInterferZone (1,OFF);          #Disables interference zone 1
End;
```

20.2 SetInterferTool

- **Function**: Switches the monitoring object.
- **Description**: This instruction switches the monitoring object, and can be directly called in the program. This instruction is a blocking type instruction.
- **Format**: SetInterferTool (id).
- **Parameter:** id indicates the monitoring object number, ranging 0 to 15.
- **Range**: It can only be used in the main task.
- **Example**:

```
Start;
  Movj P[1],V[15],Z[0],Tool[0];
  SetInterferTool (1);          #Switches to monitoring object 1
  Movj P[2],V[15],Z[0],Tool[0];
  SetInterferTool (2);          #Switches to monitoring object 2
End;
```

21 Multi-Point Recipe-Related Instructions

21.1 LoadPoints

- **Function**: Loads the global point file.
- **Description**: Reads the specified point data into the robot memory.
- **Format**: LoadPoints ("file name").
- **Parameter**:

Supports String type strings, local str type user-defined variables, Str variables, and global str user-defined variables.

File name: Specifies the name of the point file to be read into the robot memory. The extension is fixed to ".pts". It must be a point file in the currently loaded project, and cannot be a point file in specified path.

> NOTE
>
> The LoadPoints instruction is used to switch the point file flexibly when the robot program is running. After it is executed, the data of the currently loaded point file is read into the controller memory and used for the subsequent motion. Note:
>
> - An error occurs when the specified file cannot be found.
> - Whenever the robot performs project synchronization or returns to the starting line, if the point file is switched to a non-P.pts point file, the global point file is switched to "P.pts" by default.
> - This instruction can only be used in the main task.
> - This instruction overrides the global P points in the robot memory. Ensure that important points are saved before using this instruction.

- **Example**:

```
Start;
#View the P[0] point in P.pts file. The point data is consistent with the P.pts file.
Print P[0];
#Modify P[0]
P[0]= {1,2,3,4,5,6;1,1,1,1;0,0,0,0,0,0};
#View the P[0] again. The point data is consistent with the modified point.
Print P[0];
#Load P1.pts file
LoadPoints("P1.pts");
#View P[0]. The point data is consistent with the P1.pts file.
Print P[0];
End;
```

21.2 GetCurPointsFileName

- **Function**: Gets the name of the point file currently loaded by the robot.

- **Description**: After this instruction is used, the name of the point file currently loaded by the robot will be returned.
- **Format**: String xxx = GetCurPointsFileName ().
- **Parameter**: /.
- **Return value**: String type, the name of the point file currently loaded by the robot.

> NOTE
>
> Using the GetCurPointsFileName instruction in the robot program allows you to flexibly get the name of the point file that is currently loaded. It facilitates the implementation of some logic in the program. Note that:
>
> If you have not used the LoadPoints instruction to load the point file, the name of the point file obtained by this instruction will be "P.pts" by default.

- **Example**:

```
Start;
    #Move to point P[0] in the P.pts point file
    Movj P[0],V[30],Z[0],Tool[0],Wobj[0];
    #Read the point data in the calibration point file Calib.pts into the robot
    LoadPoints ("Calib.pts");
    #Move to point P[0] in the Calib.pts point file
    Movj P[0],V[30],Z[0],Tool[0],Wobj[0];
    #Get the name of the point file currently loaded by the robot
    String name;
    name = GetCurPointsFileName ();
    Print name;
End;
```

22 Safety I/O Drop-Protection Instructions

22.1 GrabClose

- **Function**: Closes the gripper.
- **Format**: GrabClose (ObjNo).
- **Range**: It cannot be used in the multi-tasking.
- **Description**: It controls the closing of the gripper.
- **Parameter:** Safety I/O instruction ID, range 1 to 16.
- **Example**:

```
Start;
  GrabClose(1);
End;
```

22.2 GrabOpen

- **Function**: Opens the gripper.
- **Format**: GrabOpen (ObjNo).
- **Range**: It cannot be used in the multi-tasking.
- **Description**: It controls the opening of the gripper.
- **Parameter:** Safety I/O instruction ID, range 1 to 16.
- **Example**:

```
Start;
  GrabOpen(1);
End;
```

23 Digital User-Defined Log-Related Instructions

23.1 SetUserLogPath

- **Function**: Sets the user log name and enables the user log writing function.
- **Format**: SetUserLogPath (strFileName, strListHead, MaxLine).
- **Parameter**:

strFileName: Type: String or Str[x]. Base file name for user logs. Actual file name: Time_strFileName. Example: 20230512132759995_SaveFile.csv.

The strFileName must be less than 20 characters in length and consist of numbers, letters, and underscores (numbers can be the first character). Example: "Filename.csv". The file save path is fixed: ~/RobotUserLog/.

strListHead: String type data, table header. Type: String or Str[x]; Optional parameter. Used to set the first line of each file, supports input of up to 256 characters. Excess characters will be discarded.

MaxLine: Type: Integer (BR variable and Int). Value range: 50 to 60000. Number of lines per user log file. Each time the buffer data reaches or exceeds the MaxLine value, a user log file is generated with MaxLine rows of data. The generated log file is named in the format of "CurrentTime_strFileName".

- **Range**: It can be used in the multi-tasking.
- **Example**:

```
## User log file name is XXXXX_T1_6.csv, stored to user log every 500 entries
##Header "Time, Torque 1, Torque 2, Torque 3, Torque 4, Torque 5, Torque 6"
SetUserLogPath("T1_6.csv","Time, Torque1, Torque2, Torque3, Torque4, Torque5, Torque6",500)
For B[0]=1,B[0]<7,Step[1]
   D[B[0]]=GetTorque(B[0]);## Get torque values of each axis
EndFor;
##Store torque values of each axis to Str[0]
SPrintf(Str[0],"J1 axis torque %.3f, J2 axis torque %.3f, J3 axis torque %.3f, J4 axis torque %.3f, J5 axis torque %.3f, J6 axis torque %.3f",D[1],D[2],D[3],D[4],D[5],D[6]);
##Store Str[0] content into buffer
Int VRet = UserLogOut(20,Time,Str[0]);
```

23.2 UserLogOut

- **Function**: Logs user data to the buffer.
- **Format**: Int Val = UserLogOut(Frequency, Time, strText1, strText2, strText3… …).
- **Parameter**:

Frequency: Type: Integer (BR variable and Int); Sets blocking time, ranging 10 to 10000 (ms) || -10 to -10000 (ms), default 20 ms. When set to a positive value: The system compares the time difference with the previous call of this instruction. If the time difference is greater than or equal to the last set Frequency value, the instruction is logged into the buffer. If the time difference is smaller than the last set Frequency value, the current instruction is skipped and execution proceeds to subsequent instructions.

When set to a negative value, the value represents the blocking time. When executing this instruction, the system pauses execution, with a blocking duration equal to the previous Frequency, and will not proceed until the save to the buffer is completed.

Time: Optional parameter. When this parameter is used, the time is automatically prefixed to the string. The default time format is: Year/Month/Day Hour:Minute:Second, e.g., "2023/05/05 13:20:16 ". A comma is added for separation during concatenation.

strTextX: Type: String or Str[x]; at least one string is required. It represents the string stored in the buffer. It can be multiple strings, which will be concatenated into a single string. It supports up to 6 strings. The total length of all strings must not exceed 200 characters. If it exceeds 200 characters, the extra data will be truncated and discarded. Each call of this instruction writes one line of data (ending with \n).

- **Return Value**: Val: Type: Integer (BR variables and Int). Return value: 0 indicates normal write; Non-zero indicates exception, including:

1. Parameter passing error.
2. Buffer exceeded 1000 lines of data, buffer is full.
3. The save path is not set, meaning the SetUserLogPath instruction has not been called in advance to set the save path.

- **Range**: It can be used in the multi-tasking.
- **Example**:

```
## User log file name is XXXXX_T1_6.csv, stored to user log every 500 entries

##Header "Time, Torque 1, Torque 2, Torque 3, Torque 4, Torque 5, Torque 6"

SetUserLogPath("T1_6.csv","Time, Torque1, Torque2, Torque3, Torque4, Torque5, Torque6",500)

For B[0]=1,B[0]<7,Step[1]

   D[B[0]]=GetTorque(B[0]);## Get torque values of each axis

EndFor;

##Store torque values of each axis to Str[0]

SPrintf(Str[0],"J1 axis torque %.3f, J2 axis torque %.3f, J3 axis torque %.3f, J4 axis torque %.3f, J5 axis torque %.3f, J6 axis torque %.3f",D[1],D[2],D[3],D[4],D[5],D[6])

##Store Str[0] content into buffer

Int VRet = UserLogOut(20,Time,Str[0]);
```

24 Interrupt Function-Related Instructions

24.1 IConnect

- **Function**: Assigns an interrupt ID to an interrupt program.
- **Description**: Associates and saves the interrupt ID with the defined callback function. When an interrupt occurs, the corresponding callback function can be easily queried via the interrupt ID.
- **Format**: IConnect IrqId, TrapFunc().
- **Parameter**:

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

TrapFunc(): Interrupt callback function; Attribute: String; Length: 1 to 32.

- **Range**: It can be used in multitasking.

24.2 IActive

- **Function**: Activates the specified interrupt.
- **Description**: Activates the interrupt task with the specified number in the current task.
- **Format**: IActive IrqId.
- **Parameter**:

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

- **Range**: It can be used in multitasking.

24.3 IDeactive

- **Function**: Deactivates the specified interrupt.
- **Description**: Deactivates the interrupt task with the specified number in the current task.
- **Format**: IDeactive IrqId;
- **Parameter**:

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

- **Range**: It can be used in multitasking.

> NOTE
>
> When an interrupt is deactivated, in addition to not responding to the interrupt when it is triggered again, tasks in the interrupt queue will also be cleared, except for those currently being executed.

24.4 IEnable

- **Function**: Enables all interrupts for the current task.
- **Description**: This instruction serves as the master switch to activate interrupts for the current task.
- **Format**: IEnable.
- **Range**: It can be used in multitasking.

24.5 IDisable

- **Function**: Disables all interrupts for the current task.
- **Description**: This instruction serves as the master switch to deactivate interrupts for the current task.
- **Format**: IDisable.
- **Range**: It can be used in multitasking.

> NOTE
>
> In addition to not responding to all interrupt signals, tasks in the interrupt queue will also be cleared, except for those currently being executed.

24.6 IDelete

- **Function**: Deletes the specified interrupt configuration, including the created signals and associated callback functions.
- **Description**: This instruction is used to delete the interrupt of the specified number in the current task.
- **Format**: IDelete IrqId.
- **Parameter**:

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

- **Range**: It can be used in multitasking.

> NOTE
>
> - In addition to deleting the configuration in the interrupt vector table, the triggered interrupt tasks in the interrupt queue will also be cleared, except for those currently being executed.
> - Except for the IDelete instruction, all other instructions will not clear the configuration information of the instruction. or delete interrupt tasks that have been stored in the buffer queue but not yet executed.

24.7 ISigIn

- **Function**: Enables interrupt instruction for input signals.
- **Description**: Associates the In signal to the interrupt ID in the current task, and sets the trigger count and trigger mode.
- **Format**: ISigIn(ONCE,IrqId, IONum, ON/OFF);/ ISigIn(IrqId, IONum, ON/OFF).
- **Parameter**:

ONCE (optional): Single trigger.

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

IONum: Interrupt-triggering I/O number; Attribute: Int; Range: Standard I/O and Bus I/O.

ON/OFF: Rising edge trigger/falling edge trigger; Attribute: Keyword.

- **Range**: It can be used in multitasking.

NOTE

- When triggered by a high level, the original level signal should remain low, and the generated high-level signal should last for 10 ms for this interrupt signal to be properly captured.
- When triggered by a low level, the original level signal should remain high, and the generated low level signal should last for 10 ms for this interrupt signal to be properly captured.
- If an interrupt is triggered during the execution of a motion instruction, the currently executing motion instruction will be stopped and discarded. If the I/O of this motion instruction has not yet been triggered, it will not be triggered subsequently.

24.8 ISigOut

- **Function**: Enables interrupt instruction for output signals.
- **Description**: Associates the Out signal to the interrupt ID in the current task, and sets the trigger count and trigger mode.
- **Format**: ISigOut(ONCE,IrqId, IONum, ON/OFF);/ ISigOut(IrqId, IONum, ON/OFF).
- **Parameter**:

ONCE (optional): Single trigger.

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

IONum: Interrupt-triggering I/O number; Attribute: Int; Range: Standard I/O and Bus I/O.

ON/OFF: Rising edge trigger/falling edge trigger; Attribute: Keyword.

- **Range**: It can be used in multitasking.

NOTE

- When triggered by a high level, the original level signal should remain low, and the generated high-level signal should last for 10 ms for this interrupt signal to be properly captured.
- When triggered by a low level, the original level signal should remain high, and the generated low level signal should last for 10 ms for this interrupt signal to be properly captured.
- If an interrupt is triggered during the execution of a motion instruction, the currently executing motion instruction will be stopped and discarded. If the I/O of this motion instruction has not yet been triggered, it will not be triggered subsequently.

24.9 ISigAD

- **Function**: Enables interrupt instruction for input signals of analog signals.
  Description: Associates the AD signal to the interrupt ID in the current task, and sets the trigger count and trigger mode.
- **Format**: ISigAD(ONCE,IrqId, ADNum,Condition,HighValue,LowValue,DeltaValue).
- **Parameter**:

ONCE (optional): Single trigger.

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

ADNum: AD number; Attribute: Int; Range: 0 to 15 ‖ 64 to 79.

Condition: Trigger condition; Attribute: Keyword ‖ or Int variable; Range as shown in the table below:

Value	Symbol	Remarks
0	ABOVE_HIGH	Above specified high value
1	BELOW_HIGH	Below specified high value
2	ABOVE_LOW	Above specified low value
3	BELOW_LOW	Below specified low value
4	BETWEEN_A _B	Between specified high value and low value
5	OUTSIDE_A_B	Outside the range of low value and high value
6	ALWAYS_SIGNAL	Always generate signal

HighValue: Interrupt triggered when above HighValue; Attribute: double ‖ int; Range: -x to +y.

LowValue: Interrupt triggered when below HighValue; Attribute: double ‖ int; Range: -x to +y.

DeltaValue: Defines the minimum logic signal difference before generating a new interrupt. Attribute: double ‖ int; Range: 0.1 to y+x.

x, y are related to the actual AD/DA module configuration.

Range: It can be used in multitasking.

NOTE

- HighValue > LowValue.
- Sample the current signal every 10 ms. An interrupt will be triggered only if: a. The absolute value of the current sample minus the sample value at the last interrupt exceeds DeltaValue; AND b. The trigger condition (Condition) is met based on the trigger mode. Otherwise, no interrupt will be triggered.
- The HighValue and LowValue parameters should be within the logical maximum and minimum values defined for the signal.
- When the Condition parameter is (BETWEEN_A _B) between high value and low value, HighValue - LowValue > DeltaValue.

Current specification:

- Above specified high value: Includes boundary value.
- Below specified high value: Excludes boundary value.
- Above specified low value: Includes boundary value.
- Below specified low value: Excludes boundary value.
- Between high value and low value: Includes boundary value.

- Outside of low value and high value: Excludes boundary value.
- Always generate signal: Excludes boundary value as long as the signal change is greater than DeltaValue.

24.10 ISigDA

- **Function**: Enables interrupt instruction for output signals of analog signals.
- **Description**: Associates the DA signal to the interrupt ID in the current task, and sets the trigger count and trigger mode.
- **Format**: ISigDA(ONCE,IrqId, DANum,Condition,HighValue,LowValue,DeltaValue).
- **Parameter**:

ONCE (optional): Single trigger.

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

DANum: DA number; Attribute: Int; Range: 0!15 ‖ 64 to 79.

Condition: Trigger condition; Attribute: Keyword ‖ or Int variable; Range as shown in the table below:

Value	Symbol	Remarks
0	ABOVE_HIGH	Above specified high value
1	BELOW_HIGH	Below specified high value
2	ABOVE_LOW	Above specified low value
3	BELOW_LOW	Below specified low value
4	BETWEEN_A _B	Between specified high value and low value
5	OUTSIDE_A_B	Outside the range of low value and high value
6	ALWAYS_SIGNAL	Always generate signal

HighValue: Interrupt triggered when above HighValue; Attribute: double ‖ int; Range: -x to +y.

LowValue: Interrupt triggered when below HighValue; Attribute: double ‖ int; Range: -x to +y.

DeltaValue: Defines the minimum logic signal difference before generating a new interrupt. Attribute: double ‖ int; Range: 0.1 to y+x.

x, y are related to the actual AD/DA module configuration.

Range: It can be used in multitasking.

NOTE

- HighValue > LowValue.
- Sample the current signal every 10 ms. An interrupt will be triggered only if: a. The absolute value of the current sample minus the sample value at the last interrupt exceeds DeltaValue; AND b. The trigger condition (Condition) is met based on the trigger mode. Otherwise, no interrupt will be triggered.
- The HighValue and LowValue parameters should be within the logical maximum and minimum values defined for the signal.
- When the Condition parameter is (BETWEEN_A _B) between high value and low value, HighValue - LowValue > DeltaValue.

Current specification:

- Above specified high value: Includes boundary value.
- Below specified high value: Excludes boundary value.
- Above specified low value: Includes boundary value.
- Below specified low value: Excludes boundary value.
- Between high value and low value: Includes boundary value.
- Outside of low value and high value: Excludes boundary value.
- Always generate signal: Excludes boundary value as long as the signal change is greater than DeltaValue.

24.11 ISigTimer

- **Function**: Enables interrupt instruction for timer signal.
- **Description**: Associates the Timer signal to the interrupt ID in the current task, and sets the trigger count and trigger time.
- **Format**: ISigTimer(ONCE, IrqId,Time).
- **Parameter**:

ONCE (optional): Single trigger.

IrqId: Interrupt ID; Attribute: Int; Range: 0 to 127.

Time: Timer interrupt interval; Attribute: double; Range: 0.1 to 1000000.0 (s).

- **Range**: It can be used in multitasking.

24.12 StopMoving

- **Function**: Stops the currently executing motion and returns the current stopping point of the robot.
- **Description**: Stops the ongoing motion trajectory, returns the current stopping point, and retains the original trajectory information. Multiple calls will not take effect.
- **Format**: JP[*] = StopMoving().
- **Range**: It cannot be used in multitasking and can only be used within an interrupt function.

NOTE

- This instruction can only be used in interrupt functions. If used in non-interrupt functions, a runtime error will occur.
- After this instruction is executed, the original motion will be stopped, and the trajectory information will be retained.
- To start new motion, during the interrupt exit sequence, you need to return to the stop point where this instruction was called.
- When entering the interrupt, if the instruction that triggered the interrupt has already completed its operation, no trajectory recovery is required upon exiting the interrupt.
- After exiting an interrupt, if there is no motion trajectory, pausing axis motion at this point will cause trajectory deviation, making trajectory recovery impossible.
- When StopMoving is called multiple times, if the previous trajectory has not been recovered, multiple calls to this instruction will not take effect.

25 System Diagnosis-Related Instructions

25.1 SetDiagnosisItem

- **Function**: Sets system diagnosis startup parameters.
- **Format**: SetDiagnosisItem(Key, Flag).
- **Parameter**:

Key: Diagnostic object keyword: All, System, Logic, Trajectory.

Flag: Check status: 0-unchecked, 1-checked; Supports using B, LB, R, LR variables, constants, and user-defined variables (int/Byte) as parameters.

- **Range**: It can be used in multitasking.
- **Example**:

```
Start;
SetDiagnosisItem(All, 1);
SetDiagnosisItem(Logic, 0);
End;
```

25.2 GetDiagnosisItem

- **Function**: Reads system diagnosis startup parameters.
- **Format**: GetDiagnosisItem(Key, Flag).
- **Parameter**:

Key: Diagnostic object keyword: All, System, Logic, Trajectory.

Flag: Variable for storing the checked state, supports B, LB, R, LR variables and user-defined variables (int/byte); 0-unchecked, 1-checked, 2-partially checked (keyword is ALL).

- **Range**: It can be used in multitasking.

- When generating system diagnosis is started, the prompt message displays: "System diagnostic file generation in progress. Do not power off."
- When system diagnosis is completed, the prompt message displays: "Prompt: System information saved successfully!".
- If an error occurs, the system displays the corresponding error message and trigger an alarm to halt the project operation.
- When a save operation is in progress and retriggered, the system displays the message "System diagnosis [Save] operation in progress, please try again later."
- Triggering system diagnosis will generate a new alarm to prevent activation during motion and avoid abnormal errors.

- **Example**:

```
Start;
GetDiagnosisItem(All, B[1]);
Print B[1];
End;
```

25.3 SetDiagnosisSave

- **Function**: Starts system diagnosis.
- **Format**: SetDiagnosisSave.
- **Range**: It can be used in multitasking.

> CAUTION
>
> The system does not automatically export diagnostic reports upon completion. Export the diagnostic report through the interface.

- **Example**:

```
Start;
SetDiagnosisItem(All, 1);
SetDiagnosisSave;
End;
```

25.4 GetDiagnosisSave

- **Function**: Gets the progress of the system diagnosis process.
- **Format**: GetDiagnosisSave(Flag).
- **Parameter**:

Flag: Variable storing the execution status of the system diagnosis, supports B, LB, R, LR variables, and user-defined variables (int/Byte). The progress value of the system diagnosis process ranges from 0 to 100.

- **Range**: It can be used in multitasking.
- **Example**:

```
Start;
SetDiagnosisSave;
GetDiagnosisSave(B[1]);
Print B[1];
End;
```

26 Bus-Related Instructions

26.1 SetIOBusState

- **Function**: Sets the slave station status management of the master bus, enabling or disabling the specified slave station on the master bus.
- **Description**:

1. Enable status of slave: The EtherCAT master normally monitors and exchanges data; if a slave status is incorrect or offline, the alarm will be triggered.
2. Disable status of slave: Shield slave status error/offline alarm, stop data exchange.

- **Format**: SetIOBusState(BusID, SlaveID, SlaveStatus).
- **Parameter**:

BusID: Keywords: EtherCAT, EtherNet/IP.

SlaveID: System variables B, LB, R, LR, custom int variables (range: EtherCAT: 1 to 16; EtherNet/IP: 1 to 8).

SlaveStatus: Supports keywords ON/OFF, constants, system variables B, LB, R, LR, custom int variables (Range: EtherCAT, EtherNet/IP: 0-Disable, 1-Enable).

- **Range**: It can be used in main task.
- **Example**:

```
Start;
  SetIOBusState(Ecat, 1, OFF);
End;
```

26.2 GetIOBusState

- **Function**: Reads the slave station status management of the master bus, and the enable/diable status of the specified slave station on the master bus.
- **Description**:

1. Enable status of slave: The EtherCAT master normally monitors and exchanges data; if a slave status is incorrect or offline, the alarm will be triggered.
2. Disable status of slave: Shield slave status error/offline alarm, stop data exchange.

- **Format**: GetIOBusState(BusID, SlaveID, SlaveStatus).
- **Parameter**:

BusID: Keywords: EtherCAT, EtherNet/IP.

SlaveID: System variables B, LB, R, LR, custom int variables (range: EtherCAT: 1 to 16; EtherNet/IP: 1 to 8).

SlaveStatus: Supports system variables B, LB, R, LR, custom int/byte/bool (Range: EtherCAT, EtherNet/IP: 0-Disable, 1-Enable).

- **Range**: It can be used in multitasking.
- **Example**:

```
Start;
  GetIOBusState(Ecat, 1, B[1]);
```

```
End;
```

27 Application Cases

27.1 Communication Settings

Vision instructions are commands used by the controller to communicate with external vision devices, either through a network or serial port.

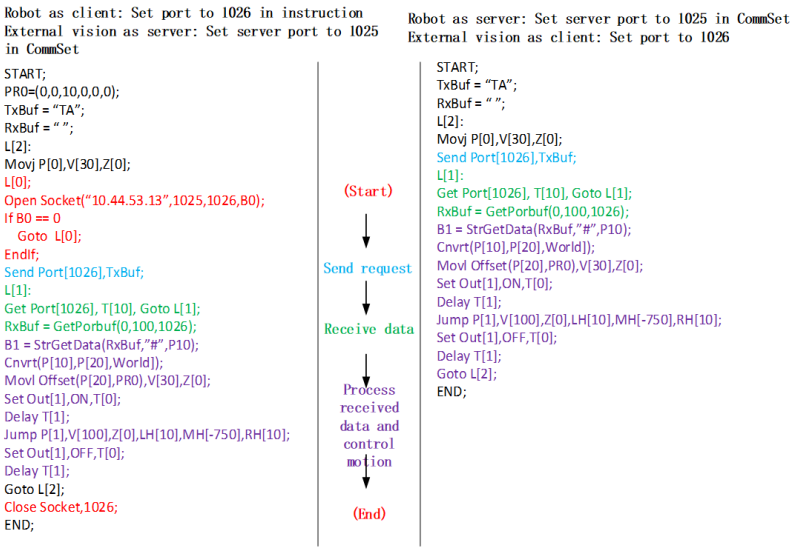
There are three forms of network port communication, depending on the usage.

(1) A local controller is used as a server and an external device is used as a client:

The robot controller acts as the server and is enabled by default upon system startup. However, if you need to set a specific port number and establish a connection on the vision device according to this port number, you can configure it in "Tool" > "Communication manager" or use the Open Server instruction in the program.

(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. Use the Open Socket instruction in the program to specify the IP and port number for connection.



(3) A local controller is used both as a client and a server:

In this case, it is equivalent to simultaneously implementing Scenario 1 and Scenario 2 under one project. Note that when using vision communication instruction, the instruction must specify whether the robot acts as a client or server through ClientType/ServerType. Example:

```
Start;
String TxBuf = "TA";
String RxBuf = "";
B[0] = 0;
   B[1] = 0;
While B[0] <> 1
##The robot acts as the client, and the external vision acts as the server. The local client port number is 1208.
      Open Socket("10.45.151.100", 2025, 1208, circle, B[0]);
   EndWhile;
While B[1] <> 1
```

```
##The robot acts as the server, and the external vision acts as the client. The external client port number is 1208.
    Open Server(2025, Circle, B[1]);
EndWhile;
Delay T[3];
Send Port[1208], ClientType, TxBuf;    #Robot acts as client to send data
Send Port[1208], ServerType, TxBuf;    #Robot acts as server to send data
L[1]:
##  Robot acts as client to receive data
Get Port[1208], ClientType, T[10], Goto L[2], R[0];
RxBuf = GetPortBuf(0, R[0], 1208, ClientType);
L[2]:
##  Robot acts as server to receive data
Get Port[1208], ServerType, T[10], Goto L[1], R[1];
RxBuf = GetPortBuf(0, R[0], 1208, ServerType);
Close Socket, 1208, ClientType;    # Robot acts as client to close connection
Close Socket, 1208, ServerType;    # Robot acts as server to close connection
```

For the serial port communication, connect the serial port and program the robot directly. The following is a programming example.

```
START;
B[0] =0;
While  B[0] <> 1
Open Com(1,19200,B[0]);
EndWhile;
Send Port[1],"TA",String;
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 tracking 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(Open

Socket/Open Server) > CnvVision(ON) > GetCnvObject > Movl P > CnvVision(OFF) > Close. A simple programming example is as follows:

```
Start;
##Open the port. The external device acts as the server, with an IP 10.45.151.108 and a port 2000.
##The local controller acts as a client and the port number is 1234.
R[0] =1234;
B[0] =0;
While B[0] <> 1
Open Socket("10.45.151.108",2000,R[0],Single,B[0]);
EndWhile;
CnvVision(Conveyor[1],OFF,R[0]);
CnvVision(Conveyor[1],ON,R[0]);                ##Opens the vision port of conveyor 1, client port number 1234
P[30]=(0,0,10,0,0,0),(0,0,0,0),(0,0,0,0,0,0); ##Defines P[30] as a point in the object frame, with coordinates (0,0,10,0,0,0)
L[0]:
Movj P[0],V[100],Z[0],Tool[0],Wobj[0];         ##Moves to the wait position
L[1]:
GetCnvObject(1,0), Goto L[1];                  ##Queries the queue for object of type 0 on conveyor #1
RefSys Conveyor(1,Tool[2]);                    ##Switches the motion reference frame to the object that has just been found
Movl P[30],V[100],Z[1],Tool[2],Cnv;             ##Moves to the P[30] in the object frame by using tool 2
Delay T[1];                                    ##Keeps the robot moving with the conveyor for 1s
Set Out[1],ON;                                 ##Turns on the switch to suck the object
RefSys Base;                                   ##Switches the motion reference system to the robot
Jump P[1],V[100],Z[0],Tool[0],Wobj[0],LH[10],MH[-750],RH[10];  ##Moves the object to P[1]
Set Out[1],OFF;                                ##Places the object
Goto L[0];
CnvVision(Conveyor[1],OFF);
Close Socket,R[0];
End;
```

Note: When the robot acts as client and server simultaneously, if the CnvVision instruction is used, the port number must also be specified via ClientType/ServerType to determine whether the vision port being opened corresponds to the robot acting as a client or as a server.

27.2 Interrupt Example

#1. Without motion

```
Start;
  Int IrqId = 0;     ## Interrupt ID
```

```
  Int IONum = 50;    ## Interrupt signal I/O number
  IDelete IrqId;   ## Delete interrupt and clear configuration corresponding to interrupt ID
  IConnect IrqId, TrapFunc();  ## Bind callback function TrapFunc to interrupt ID
  ISigIn (IrqId, IONum, ON);   ## Bind I/O signal and set trigger condition as rising edge trigger
  ## User processing logic...
  While 1
    Delay T[1];
  EndWhile;
End;
## Callback function
Trap TrapFunc()
  Print "hello";
EndTrap;

#2. With motion
Start;
  Int IrqId = 0;    ## Interrupt ID
  Int IONum = 50;    ## Interrupt signal I/O number
  IDelete IrqId;   ## Delete interrupt and clear configuration corresponding to interrupt ID
  IConnect IrqId, TrapFunc();  ## Bind callback function TrapFunc to interrupt ID
  ISigIn (IrqId, IONum, ON);   ## Bind I/O signal and set trigger condition as rising edge trigger
  While 1
    MovAbsJ JP[0],V[30],Z[0],Tool[0], Until IN[0]==ON;
    MovAbsJ JP[1],V[30],Z[0],Tool[0], Until IN[0]==ON;
  EndWhile;
End;
Trap TrapFunc()
JP[100] = StopMoving;    ## Stop current motion and record the stopping point
  MovAbsJ JP[1],V[30],Z[0], Tool[0];  ##Execute new motion
  ## Other processing logic
  ## ...
  ## Other processing logic
  MovAbsJ JP[100],V[30],Z[0], Tool[0];    ##Return to stopping point
EndTrap;

#3. Define the callback function in the public module
```

```
#####main.pro
Include "moduleTrap.pro"
Start;
   Int IrqId = 0;      ## Interrupt ID
   Int IONum = 50;     ## Interrupt signal I/O number
   IDelete IrqId;    ## Delete interrupt and clear configuration corresponding to interrupt ID
   IConnect IrqId, moduleTrap .TrapFunc(); ## Bind callback function moduleTrap .TrapFunc to interrupt ID
   ISigIn (IrqId, IONum, ON);    ## Bind I/O signal and set trigger condition as rising edge trigger
   ## User processing logic...
   While 1
      Delay T[1];
   EndWhile;
End;

#####moduleTrap.pro
## Callback function
Trap TrapFunc()
   Print " moduleTrap hello";
EndTrap;
```

27.3 File Read-Write Instruction Usage Examples

```
Start;
   FileHandle fd1; #Define file handle variable
   FCreateDir(FD_RC_SD, "/PosData"); #Create PosData folder in user directory
   FOpen(FD_RC_SD, FM_RW, "/PosData/RobPosition.txt", fd1); #Open RobPosition.txt file in read-write mode under controller user directory
   For R[0] = 0,R[0] < 100,Step[1]
      P[100]= GetCurPos(); #Get current position
      Str[0] = "X = " + RToStr(P[100].X) + ", Y = " + RToStr(P[100].Y) + ", Z = " + RToStr(P[100].Z); # Concatenate current position to string
      Str[0] = Str[0] +  ", A = " + RToStr(P[100].A) + ", B = " + RToStr(P[100].B) + ", C = " + RToStr(P[100].C);
      FWriteALine(fd1, Str[0]); #Write string to file
      Delay T[1]; #Delay by 1s
   EndFor;
   FClose(fd1); #Close file
   FOpen(FD_RC_SD, FM_R, "/PosData/RobPosition.txt", fd1); #Open file in read-only mode
   FReadALine(fd1, Str[1]); #Read one line of string
```

```
    While 0 <> Strcmp(Str[1], "EOF"); #Determine if file has ended
        Print Str[1]; #Print read string
        FReadALine(fd1, Str[1]); #Read next line
    EndWhile;
    FClose(fd1); #Close file
End;
```

28 Temporarily Unsupported Instructions for Virtual Controllers

Type	Instruction
Basic variable	Pallet/LPallet
Motion instruction	Movs
Signal Processing Instructions	BindTcpSpeed, CloseBindTcpSpeed
System parameter instruction	SetFlyWait, SetAccuracyMode, SLVSMode, SLDataClear, SetVarRecordMode, GetRobotType, SetJumpSLMode
Process Control Instructions	Trap
Operation instruction - value conversion	CVDataToPose
Operation instruction - coordinate calculation	LoadPointFromFile, GetAPointFromFile, WriteAPointToFile, SavePointFile, Lpallet, Pallet, P=Pallet, CVToRobPos
Communication instruction	Open Socket, Close Socket, Send Port, GetPortbuf, Open Com, Close Com, Get Port, GetPortState, GetSocketNo, SetSocketRecvType, GetAString, GetDynamicIP
Information interaction instructions	GetPlcVarByte, GetPlcVarDInt, GetModBusCoil, GetModBusReg, GetPlcVarInt, GetPlcVarLReal, SetModBusCoil, SetModBusReg
Collision detection	SetCollMode, SetAxisCollMode, SetAxisCollLevel, SetCollDist
Conveyor	CnvVision, CnvFastCatch, RefSys, GetCnvObject, CopyCnvObject, ClearCnvObject, Cnv, GetCnvObjType, GetCnvSpeed, GetCnvCntPos, GetCnvObjOutNum, CnvTrigVis, GetCnvObjData, SetCnvVisOTMode, GetCnvVisOTNum
Position latch	LatchEnable, ClearLatchPos, GetLatchPos
Temperature rise protection	GetEncoderTemper
Independent axis-related	IndCMove, IndSpeed, IndReset
Soft float	SoftModeAct, SoftModeDeact, SoftFollow
Interference zone-related	SetInterferZone, SetInterferTool
Point recipe-related	LoadPoints, GetCurPointsFileName
Safe I/O drop protection	GrabClose, GrabOpen
Digital user-defined log related	SetUserLogPath, UserLogOut
Interrupt function related	IConnect, IActive, IDeactive, IEnable, IDisable, IDelete, ISigIn, ISigOut, ISigAD, ISigDA, ISigTimer, StopMoving
Diagnosis related	SetDiagnosisItem, GetDiagnosisItem, SetDiagnosisSave, GetDiagnosisSave

Type	Instruction
Bus related	SetIOBusState, GetIOBusState

Service and Support

Should you encounter a safety accident during the use or operation of the product, or face challenges in operating and maintaining the equipment, which remain unresolved after the relevant documentation is consulted, we provide multiple channels to ensure prompt resolution:

- Channel #1: Contact service@inovance.com.
- Channel #2: Visit https://www.inovance.com/global to access document downloads, after-sales support, spare parts ordering, repair applications, and authenticity verification services.
- Channel #3: Download My Inovance app (https://zshc-eu.inovance.com/download-pc/) where you can access products info and documentation, and query product parameters.

We are committed to providing you with quick and professional technical support, and we look forward to your satisfaction and trust.

19012825A00

Shenzhen Inovance Technology Co., Ltd.

www.inovance.com

Add.: Inovance Headquarters Tower, High-tech Industrial Park,
Guanlan Street, Longhua New District,
Shenzhen 518000, P.R. China

Tel: (0755) 2979 9595 Fax: (0755) 2961 9897

Suzhou Inovance Technology Co., Ltd.

www.inovance.com

Add.: No. 52, Tian E Dang Road, Wuzhong District, 215104,
Suzhou City, Jiangsu Province, P.R. China

Tel: (0512) 6637 6666 Fax: (0512) 6285 6720