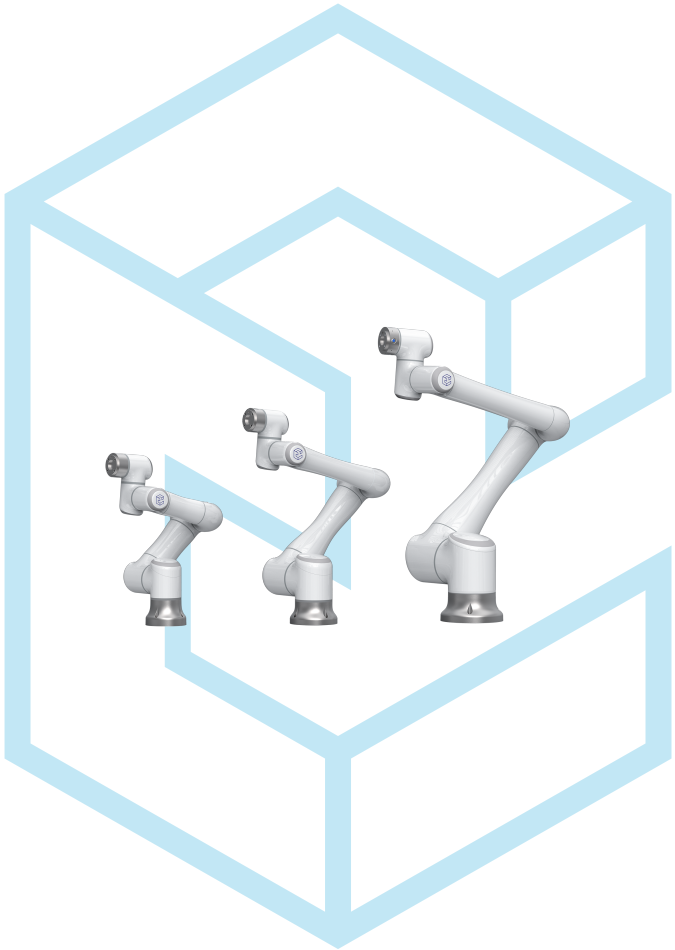
ELITE ROBOTS
艾利特机器人

ELITE ROBOTS EC Series Programming Manual



JBI Script Manual

Suzhou Elite Robot Co., Ltd

2022-12-28

Version: Ver3.8.4

Contents

1 Interactive programming via Teach Pendant **2**

2 JBI Script Language **5**

2.1 JBI script keywords 5

2.2 JBI Script Syntax 7

2.3 JBI Logic Control Statements 7

2.3.1 IF command 7

2.3.2 WHILE command 7

2.3.3 CALL and JUMP command 8

2.3.4 Pause command 9

2.3.5 Wait command 9

2.3.6 Time delay command 10

2.3.7 Cancel collision temporarily 11

2.4 Motion Instructions 11

2.4.1 Joint motion command (MOV) 11

2.4.2 Linear motion command (MOV) 13

2.4.3 Circular motion command (MOV) 14

2.4.4 Drag path re-perform command (MOV) 16

2.4.5 Micro-segment interpolation instruction (MOV) 16

2.4.6 Joint motion command (MOVE) 17

2.4.7 Linear motion command (MOVE) 18

2.4.8 Circular motion command (MOVE) 21

2.4.9 Drag path re-perform command (MOVE) 24

2.4.10 Micro-segment interpolation instruction (MOVE) 24

2.4.11 Initialize transparent transformation process 26

2.4.12 Transparent transformation start position 26

2.4.13 Transparent transformation end 26
2.4.14 Load job file 27
2.4.15 Unload job file 27
2.4.16 Division instruction 27
2.5 Mathmatic Instructions 28
2.5.1 Increment instruction 28
2.5.2 Decrement instruction 28
2.5.3 Addition instruction 28
2.5.4 Subtraction instruction 29
2.5.5 Multiplication instruction 30
2.5.6 Division instruction 31
2.5.7 Residual calculation 31
2.5.8 Bitwise and operation instruction 32
2.5.9 Bitwise or operation instruction 32
2.5.10 Opposition operation instruction 33
2.5.11 Different or operation instruction 33
2.5.12 Calculate the distance between two points. 34
2.5.13 Specify the coordinate system command 34
2.5.14 Pose Multiplication 35
2.5.15 Inverse Kinematics 35
2.5.16 Forward Kinematics 36
2.5.17 Pose inversion 36
2.5.18 Calculate the pose variation 37
2.6 Set and Get Instructions 38
2.6.1 Variable assignment instruction 38
2.6.2 Joint position assignment instruction 38
2.6.3 Spatial position assignment instruction 39
2.6.4 Get current joint position 40
2.6.5 Get a specific tool frame 40
2.6.6 Set user coordinates 40
2.6.7 Get a specific user frame 41

2.6.8 Set V variable to tool . . . 41
2.6.9 Set the current running tool number . . . 41
2.6.10 Get the current running tool number to B variable . . . 42
2.6.11 Get tcp force . . . 42
2.6.12 Get joint torque . . . 42
2.6.13 Set the payload . . . 43
2.6.14 Create user coordinate system . . . 43
2.6.15 Get the actual TCP pose . . . 44
2.6.16 Get the target interpolation TCP pose . . . 44
2.6.17 Get the current actual joint . . . 44
2.6.18 Get the current target interpolation joint . . . 45
2.6.19 Get the linear interpolation pose . . . 45
2.7 Input and Output Instructions . . . 45
2.7.1 The digital signal output instruction . . . 45
2.7.2 Read the input signal . . . 46
2.7.3 Analog signal output instruction . . . 47
2.7.4 Get the analog input port voltage . . . 47
2.7.5 Virtual signal output instruction . . . 48
2.7.6 Reads the specified M signal . . . 49
2.7.7 Pulse operation . . . 50
2.8 Other Instructions . . . 50
2.8.1 Save all P variables . . . 50
2.8.2 Print command . . . 51
2.8.3 Clear instruction . . . 51
2.8.4 Force control instruction . . . 52
2.8.4.1 Start the force control mode . . . 52
2.8.4.2 End the force control mode . . . 52
2.8.4.3 Clear all values of the force sensor to zero . . . 53

2.9 Lua Control Instruction 53
2.9.1 Run a specific script file 53
2.9.2 Stop a script file 54
2.9.3 Reload a script file 54
2.9.4 Get a script file state 54

3 JBI Script Examples 55

3.1 Subroutine call 55
3.2 Logical gate operation 55
3.3 Logical operation priority 57
3.4 Forward and inverse solutions 57
3.5 Calculation of coordinate system transformations 58

Preface

JBI (short for Job Instruction) script for Elite collaborative robots can be either interactive programming via a teach pendant or text-based editing on PC. The interactive programming method is to generate a script file in a certain format directly on the teach pendant without the need to memorize the instruction syntax and parameter types. The text-based editing method can improve the programming efficiency and is more suitable for engineers who is more familiar with JBI language. Elite robot controller uses Linux operating system and comes with sFTP server. You can access the file with extension .jbi by logging into /rbctrl folder with FTP software (e.g. FileZilla, with username :root, password:elite2014, port:22.) Some free text editors, such as notepad++, come with an sFTP plug-in for remote online editing.

Chapter 1 Interactive programming via Teach Pendant

1. Turn on the controller and switch to TEACH mode. Go to main page where all program files are listed as shown in **Figure** 1-1 . Create a new program file or open an existing one.

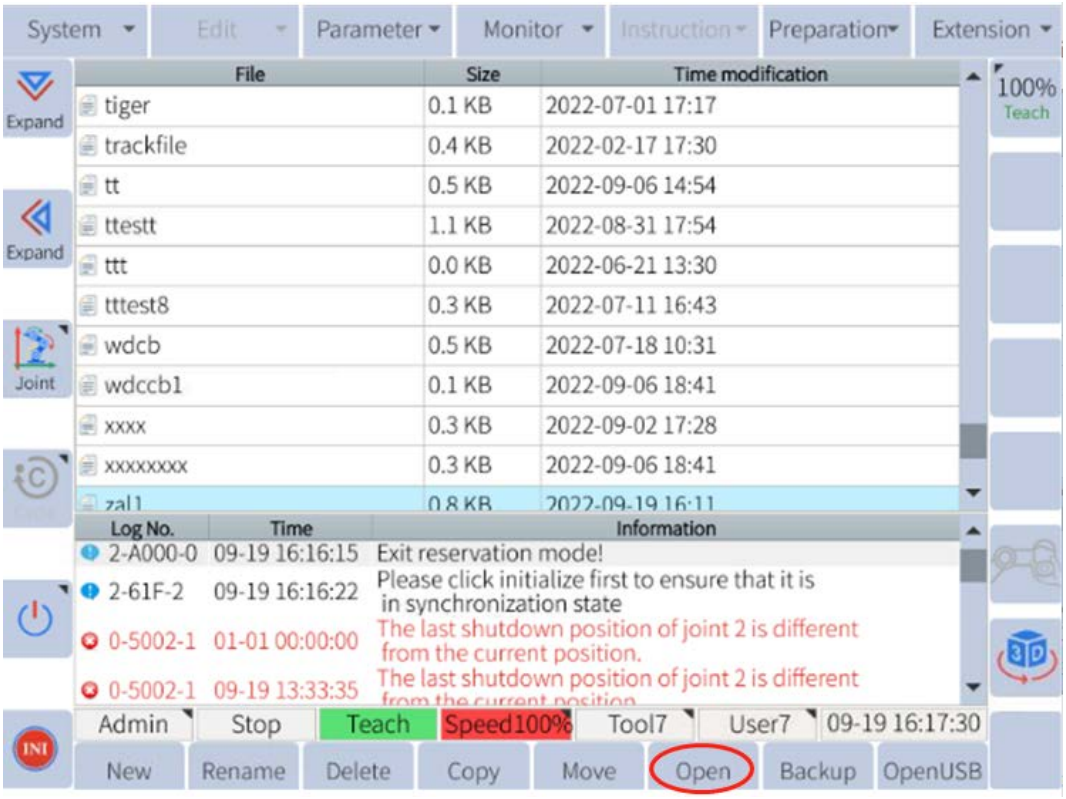


Figure 1-1 : Program list page

2. Click on the “Instruction” tab on the main menu or on the submenu bar “Custom” shortcut tab on the sub-menu bar. Select appropriate command, as shown below.

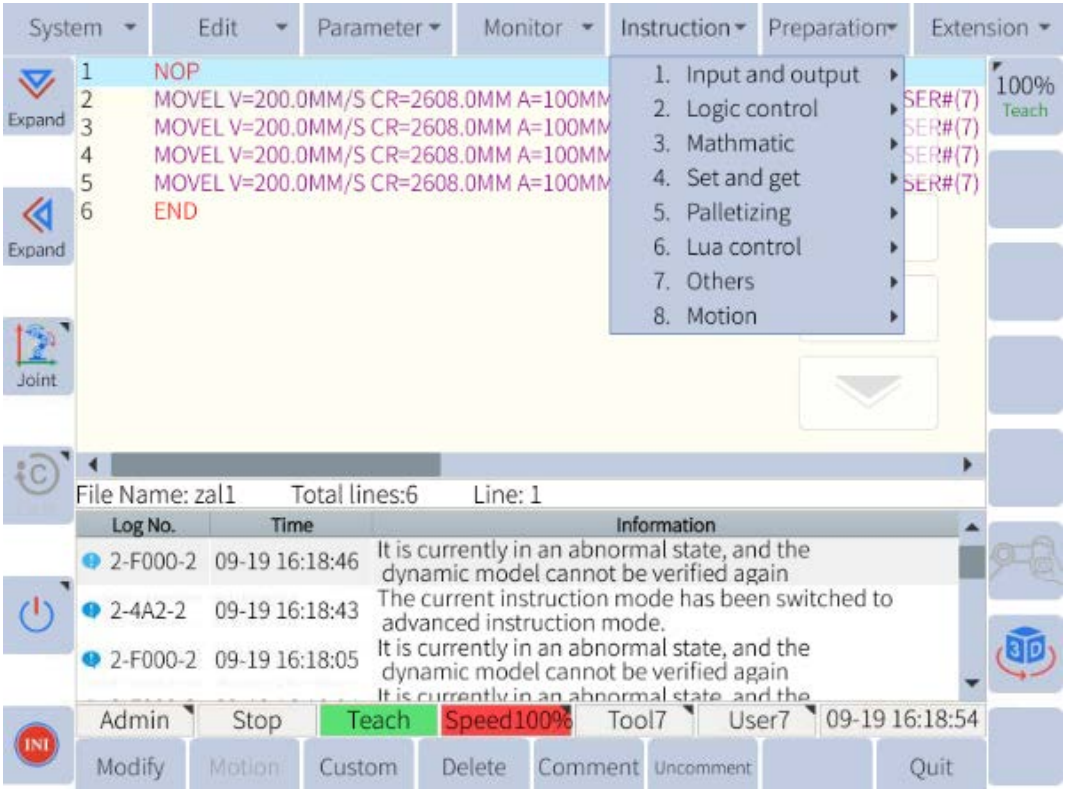


Figure 1-2 : Instructions

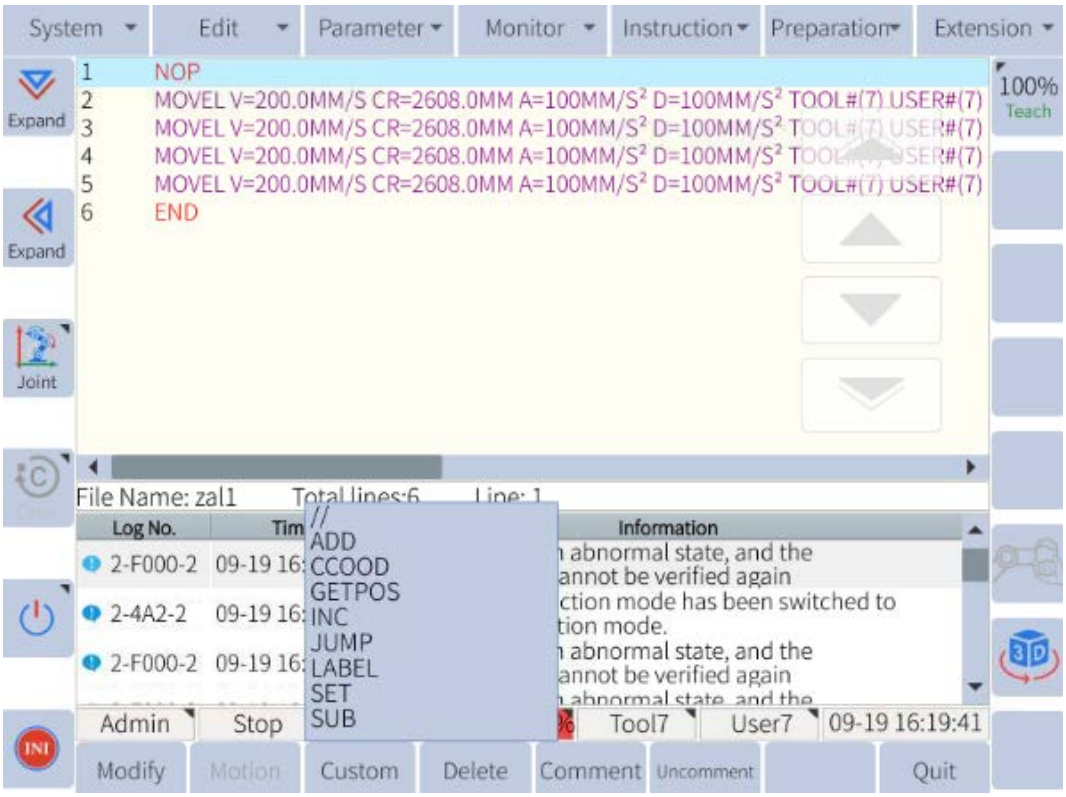


Figure 1-3 : Logic instruction

3. After correct configuration of the parameters, click ”Confirm” to insert the instruction; click ”Cancel” key to cancel the command editing. To insert or edit any motion command, please

ensure that the servo is in the excitation state (the servo switch is a three-level switch, which is fully released as a level, the middle gear is a level, and fully pressed as a level. When the switch is in the middle gear, it means the servo is in excitation state). An example of parameter configuration is shown in **Figure** 1-4 . Grayed out unit is optional, and the triangle symbol indicates the presence of a drop-down selection.

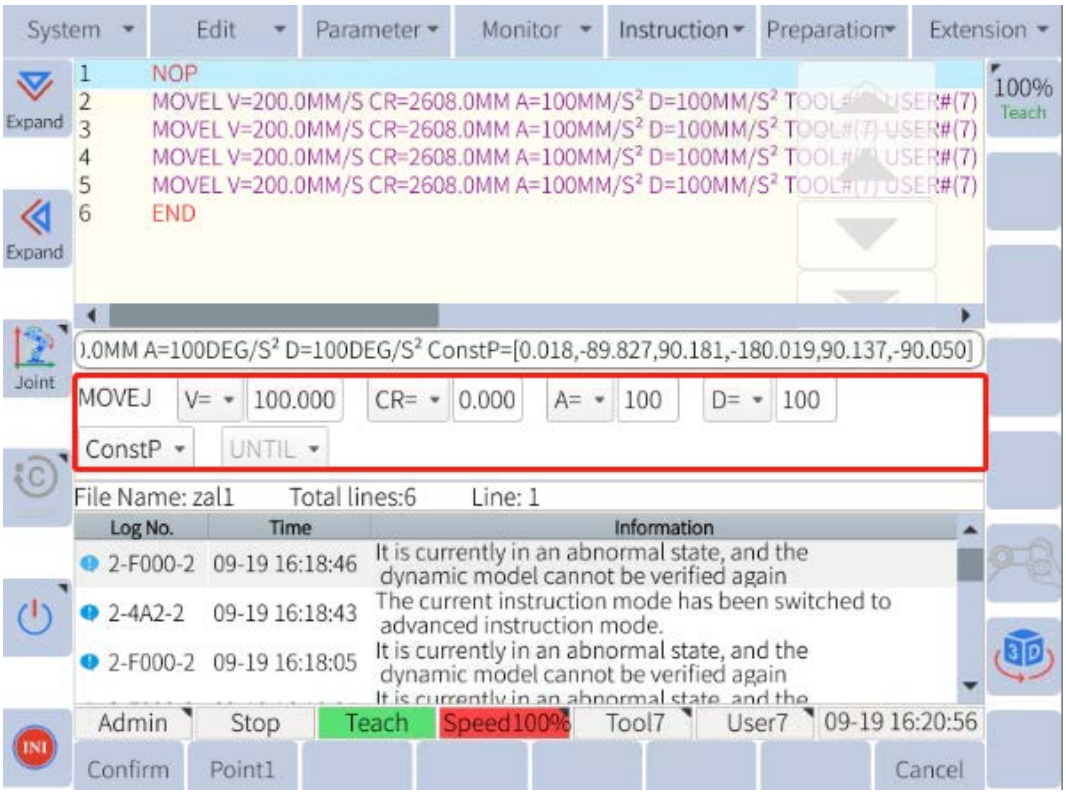


Figure 1-4 : Command parameter setting

Chapter 2 JBI Script Language

Elite collaborative robots use a specific script language called JBI (short for Job Instruction), developed independently by Elite. Unlike Lua, JBI is the only language with robot motion control instructions, including variables, logic, loops, calculation, IO monitoring, etc. JBI script can quickly share data with soft PLC and Lua programs through M registers, global variables, etc. JBI scripts are stored in a text file with a ".jbi" extension, called a JOB.

This article engages.

1. Script instruction contains script and parameter, while parameter contains parameter name and parameter value, e.g., CALL JOB:1 [IF B001=2], where "[]" means this is an optional parameter, and the italic part is command.
2. Lower cased parts are the types of the required parameter. For example, "job" means the parameter is a JOB such as JOB :1 = 1.jbi. "joint#3" means the third element of joint type variable, see Section 2.1 for details.
3. {Primary-menu}->{Secondary-menu} indicates the hierarchy of the menu on the teach pendant. "{}" indicates the description. For example, ={equal} means "= " works as "equal" here.

2.1 JBI script keywords

JBI scripts have the following data types: int{integer}, uint{positive integer}, double{floating-point}, joint{joint position}, pose{cartesian pose}, bool{boolean}, string{string}, bit{bit}.

Built-in global variables include:

1. B{uint}, positive integer variable, value range 0~2147483647, size 0~255, e.g. B001;
2. I{int}, integer variable, value range -32768~32767, size 0~255, e.g. I001;
3. D{double}, float variable, value range -1e+09~1e+09, number 0~255, e.g. D001;
4. P{joint}, joint position variable, {J1,J2,J3,J4,J5,J6,0,0}, j# corresponds to the angle value of each joint in deg(°), size 0-255,e.g. P001 .
5. V{pose}, Cartesian pose variable, x,y,z,Rx,Ry,Rz, x,y,z is the position of TCP (tool center point) in mm, Rx,Ry,Rz is the rotation vector in rad , e.g. V001. The unit vector of the rotation vector is axis of rotation and the norm is the rotation angle.

The variables B,I,D,P,V can be viewed and configured through the teach pendant {Monitor}{Global variable}. B, I, D, P, V variables are global variables, which can be accessed by variable type + serial number (for example, B001), and can also be accessed via Lua programs. Adding a prefix "L"(local) in front of each variable will make local variables, as shown in **Figure** 2-1 , such as

LB001, which cannot be accessed by Lua program. All global variables support power-down retention.

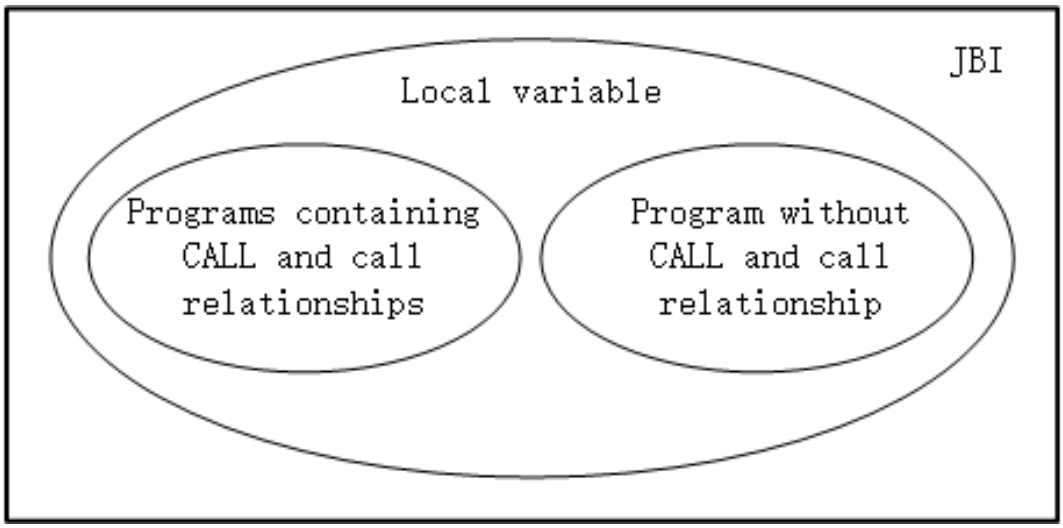


Figure 2-1 : Local variables

M{bit} virtual coil, size M000 M1535. A single M variable is a coil, while multiple consecutive M virtual coils form registers: M#(uint)1-bit register, MGH#(uint)4-bit register, MG#(uint)8-bit register, MGD#(uint)16-bit register. MGD#() is used to save the register address. Virtual input, size: M000 M399, M912 M1423. Virtual output, size: M400 M1535, M1424 M1535: M400 M527, M1472 M1535 are occupied by system for internal use; M528 M911, M1424 M1471 can be used freely by user.

- 4-bit registers with a maximum size of 1536/4=384, with the register address being the first coil address of each successive 4-bit coil.
- 8-bit registers with a maximum size of 1536/8 = 192, with the register address being the first coil address of each successive 8-bit coil.
- Modbus is a 16-bit register with maximum size of 1536/16=96 registers, and the register address is the address of the first coil of 16 successive coils.
  Note: Only M528~M1471 coils can be used by user. 16-bit registers with a size of (1471-527)/16 = 59, with the register being corresponded with a successive set of 16 M virtual coils

Logical operators: |{or},&{and}, 1=1{true}, 1=0{false};

Compare operator:={equal}, <>{not equal}, >{greater than}, <{less than}, >={greater than or equal to }, <={less than or equal to};

Digital value{switch}:ON{1}, OFF{0};

Program structure:NOP{start of program}, END{end of program};

Coordinate system{frame}:CART{Cartesian coordinates}, JOINT{joint coordinates}, USER#(){user coordinates, size: 0-7}, TOOL#(){tool coordinates, size: 0-7},

Input/Output{IO}:OT#(uint){1-bit digital output port}, OGH#(uint){4-bit digital output port}, OG#(uint){8-bit digital output port}; IN#(uint){1-bit digital input}, IGH#(uint){4-bit digital input}, IG#(uint){8-bit digital input}; AO#(uint){analog output}, AI#(uint){analog inpu}.

2.2 JBI Script Syntax

JBI language is case-sensitive, all instructions and parameters are uppercase. There is no requirement for indentation. Different instructions need to be on independent l ines. The common format for JBI program is as follows, with the instructions enclosed between the keywords NOP and END. The previous data is the joint position data automatically generated by the software and corresponds to the use of non-variable types of MOV commands such as Lines 5 and 6 of the following code.

```
1 //Storage of fixed points (//(Note)
2 C00000=0.0000,-90.0000,0.0000,0.0000,90.0000,0.0000,0.0000,0.0000
3 C00001=0.0000,-90.0000,0.0000,0.0000,90.0000,0.0000,0.0000,0.0000
4 NOP
5 MOVJ VJ=100% PL=0
6 MOVL AV=10.0 MM/S PL=0
7 END
```

2.3 JBI Logic Control Statements

2.3.1 IF command

IF-ELSEIF-ELSE Syntax.

An IF must correspond to an ENDIF, ELSEIF does not need to correspond to an ELSEIF; IF nested IF, there will be no error when the judgment statement is the same.

```
1 IF D000=0&1=1 THEN
2 // Add Code
3 ELSEIF D000=1| 1=0  THEN
4 // Add Code
5 ELSE
6 // Add Code
7 ENDIF
```

2.3.2 WHILE command

WHILE-CONTINUE-BREAK Syntax.

A WHILE corresponds to an ENDWHILE; in the single loop mode, there is only one set of WHILE judgment statements in the program. If BREAK is not included and the judgment condition is established, the program will loop continuously.

```
WHILE LB000=0 DO
// Add Code
IF D000=0 THEN
BREAK
ELSE
CONTINUE
ENDIF
ENDWHILE
// When the loop mode is single loop, the loop starts when LB000=0, and
    when D000=0, the code after the while...DO instruction and before
   the IF...THEN instruction is executed once;
//When LB000=0, the loop starts, when D000 is not equal to 0, the code
   after the while...DO instruction and before the IF...THEN
   instruction is executed multiple times
```

2.3.3 CALL and JUMP command

JUMP-LABEL jump statement, JUMP label [IF true]:

```
// LABEL The length of the tag name cannot be exceeded 31 characters.
LABEL *p1
// Add Code
// JUMP can be followed by a tag name.
JUMP *p1  IF B000=1
```

CALL-RET statement, CALL job [IF true], RET [IF true]:

```
//CALL can be followed by a program name of the program needing to
   jump.
CALL  JOB:1 IF B000=0|B002=1
```

Both JUMP and CALL support loop nesting between programs.

Both JUMP and CALL can call JOB. However, CALL calls JOB with a return statement and continues to run the main program after returning while JUMP dose not return.

Note that CALL can call up to 10 levels, and the CALL of all programs cannot exceed 126 (the one with the same name is counted as one). RET can only be used with CALL, and JUMP cannot be used with RET

The CALL instruction calls the file, and it stops after the operation ends, and the interface displays the program interface where the call instruction is located;

CAUTION

1. There are CALL instructions and local variables in program A. The CALLed program B contains local variables with the same name. The calculation result of the local variable with the same name in program B will not affect the value of the local variable with the same name in program A.
2. There is no CALL instruction in program C and program D, and they contain local variables with the same name. After running the C program, run the D program. The calculation result of the local variable with the same name in the program C will affect the value of the local variable with the same name in the program D.

2.3.4 Pause command

```
PAUSE [IF true]
```

Function: Pause command. The program will pause as soon as the robot executes this command.

Parameters: [IF true]: Optional judging condition.
The state of any variable or constant is determined by a logical operator.
Multiple conditions can be superimposed in series with the logical operators|{or} and &{and}.

Example:

```
PAUSE IF B001=2
// If the global value of variable B001 is equal to 2,
   then the program is paused at this line.
PAUSE IF OT# (003) =0 | D002>5
// If the digital output Y003 has signal value 0 or the
   value of global variables D002 is greater than 5, the
   program is paused at this line.
```

2.3.5 Wait command

```
WAIT bool [T∞=]
```

Function: Waiting for the instruction, when the condition is not established, the robot executes this instruction and waits for the time T here;

Parameters: bool: Optional waiting condition.
The condition is the judgment of the state of any variable or constant. When the condition is satisfied, it continues to run, and the robot state is running. When the condition is not met, the program waits for time T.
[T= ∞ .]: Optional maximum waiting time (unit: seconds). The time to wait when the condition is not met. If the condition is met, then no wait If no value is assigned, the default is to wait forever.

Example:

```
WAIT M# (600) =1
// wait until M#(600) signal becomes 1
WAIT D001>0.3 T=3
// wait until D001 greater than 0 or 3 seconds time out,
   continue with subsequent commands.
WAIT LI>8 & OGH# (1) =8 T=1.2
// wait until local variable LI is greater than AND Y004-
   Y007=0,0,0,1, or 1.2 seconds time out. continue with
   subsequent commands.
```

2.3.6 Time delay command

```
TIMER T=double
```

Function: Time delay command. The program stays here for the specified time while executes this command.

Parameters: T=double: Delay time, range 0~10000, accept integer or double assignment, unit is seconds (s).

Example:

```
TIMER T=3
// Delay 3 seconds
```

2.3.7 Cancel collision temporarily

```
MCWAIT bool T=double S=int
```

Function:	Turn off the collision alarm temporarily and wait until the exit conditions are met or the timeout elapses
Parameters:	bool: judgment condition. When the condition is fulfilled, the robot will not wait. It will execute the next command and start the collision test. When the condition is not fulfilled, the robot will wait and cancel the collision test temporarily. The state is "operation in progress". T: timeout S: sensitivity
Example:	`MCWAIT IN#(1)=ON`
Note:	This command is applicable to 2.15.0 and above versions.

2.4 Motion Instructions

Special notes on the Elite system movement command and the speed of the motion instructions.

1. In TEACH mode, while trying to move the robot to a tough point in the program, Elite system handles the movement in the way of MOVJ interpolation.
2. In TEACH mode, while trying to move the robot to a tough point in the program, the speed of the movement is only related to the overall speed set on the teach pendant.
3. In PLAY mode, when the speed type is VJ, V, VR of the motion instruction, the robot runs automatically at speed = (set value of speed parameter) * (overall speed (%) set on the teach pendant).
4. In PLAY mode, when the speed type is AV, AVR, the robot runs automatically at speed = t (set value of speed parameter.

For the parameters TOOL, USER and UNTIL, the subscript values are of type int and have the range. In the phase of decoding, the internal optimization will be done in the system. That is, when assigning a fractional or float variable to an integer variable, only the integer part (not rounded, but the fractional part is completely discarded) is used.

2.4.1 Joint motion command (MOV)

```
MOVJ [joint] VJ=int PL/CR=int [ACC=int] [DEC=int] [UNTIL bool]
```

Function:	Joint motion command, the robot will move by joint interpolation when executing this command
Parameters in simple mode:	int./B: Speed parameter, the range is 1%~100%, the value of speed can be directly entered as a numerical value (percentage) or set using the B variable PL/CR=int.: PL, trajectory blending level, the value range is 0~7. The larger the value, the higher the degree of trajectory fusion. When not selected, it defaults to PL=0. CR, blend radius, unit: mm. The parameter PL is deprecated.
Parameters in advanced mode:	[joint]: Joint angle variable. When there is this optional parameter, it means to use the specified P variable, if not, it means to use the fixed point taught. P variable allows manual modifications; [ACC=int]: Acceleration percentage, the range is 0~100; [DEC=int]: Deceleration percentage, the range is 0~100; [UNTIL bool]: The condition for the execution of this motion instruction, to judge a variable or constant. When the conditions are met, this point does not run, otherwise, it runs this point. Multiple conditions can be superimposed in series by logical operators |{or} and &{and}

Example:

```
MOVJ VJ=100% CR=50.0MM
//Move to the taught fixed point with joint interpolation,
   100% speed, and blend radius of 50mm
MOVJ P000 VJ=100% CR=50.0MM
// Move to the P000 point with joint interpolation, 100% speed
   , and a blending radius of 50mm
SET B000 30
MOVJ VJ=B000 CR=50.0MM
//Move to the taught fixed point with joint interpolation,
   speed of B000, and blending radius of 50mm
MOVJ P001 VJ=60% CR=0.0MM ACC=75% DEC=40% UNTIL M#(442)=ON
//At 60% speed, 75% acceleration, 40% deceleration, move to
   point P001 until the M#(442) interference area signal is
   triggered, then the movement stops
MOVJ P002 VJ=100% CR=50.0MM
//When M#(442)=ON is satisfied, end the movement of point P001
   , and move to point P002 at 100% speed
```

2.4.2 Linear motion command (MOV)

```
MOVL [joint/pose] V=int PL/CR=int [ACC=int] [DEC=int] TOOL#(int) [
   UNTIL bool]
```

Function: Linear motion command, when the robot executes this command, it will move by linear interpolation

Parameters in simple mode:
V=int: Speed parameter, which can be set as one of V, VR, AV, and AVR.
V=Linear speed, range 1~3000MM/S;
VR=Rotation angular velocity: 1~300°/S;
AV= is the absolute linear velocity: 1~3000MM/S;
AVR= is the absolute rotational angular velocity: 1~300°/S;
PL/CR=int.: PL, trajectory blending level, the value range is 0~7. The larger the value, the higher the degree of trajectory fusion. When not selected, it defaults to PL=0.
CR, blend radius, unit: mm. The parameter PL is deprecated.

Parameters in advanced mode:

[joint/pose]: Joint angle or pose variable. When there is this optional parameter, it means to use the specified P or V variable, if not, it means to use the taught fixed point. P or V variable allows manual modifications;
[ACC=int]: Acceleration percentage, the range is 0~100;
[DEC=int]: Deceleration percentage, the range is 0~100;
[TOOL#(int)]: optional parameter, tool coordinate system number, range: 0~7;
[UNTIL bool]: The condition for the execution of this motion instruction, to judge a variable or constant.
If the condition is established, the movement will be suspended immediately, and if it is not established, the movement will continue. Multiple conditions can be superimposed in

Example:

```
MOVL AV=400MM/S CR=100.0MM
//Move to the fixed teaching point position by linear
   trajectory interpolation, speed of 400.0MM/S, and blend
   radius of 100mm
MOVL P001 AV=400.0 MM/S CR=50.0MM
//Move to point P001 by linear trajectory interpolation at a
   speed of 400.0MM/S.
SET I000 100
MOVL AV=I000 CR=0.0MM
//Move to the taught fixed point with linear interpolation and
    speed I000 MM/S
MOVL AV=100MM/S CR=0.0MM UNTIL IN#(1)=1
// Move to the taught fixed point with linear interpolation
   and speed of 100MM/S until the external digital input X1 is
    valid, then the movement stops
SET I000 100
MOVL P000 AV=I000 CR=100.0MM UNTIL IN#(1)=1|M#(528)=1&D000=10.145
// Move to P000 point with linear interpolation, speed I000 MM
   /S, and blend radius of 100mm until the external
   digital input X1 is valid or the virtual output M528 is
   valid and the variable D000=10.145, then the movement stops
```

2.4.3 Circular motion command (MOV)

```
MOVC [joint/pose] V=int PL/CR=int FPT [ACC=int] [DEC=int] TOOL#(int)
   [UNTIL bool]
```

Function: Circular motion command; when the robot executes this command, it will complete a segment of circular path movement through the three points taught by circular interpolation.
Note: This command must be applied in the program in groups of 3, otherwise the arc trajectory cannot be realized and the system will report an error. In addition, when the command is single-stepped in PLAY mode, the robot moves in the way of joint interpolation.

Parameters in simple mode: V=int: Speed parameter, which can be set as one of V, VR, AV, and AVR.
V=Linear speed, range 1~3000MM/S;
VR=Rotation angular velocity: 1~300°/S;
AV= is the absolute linear velocity: 1~3000MM/S;
AVR= is the absolute rotational angular velocity: 1~300°/S;
PL/CR=int.: PL, trajectory blending level, the value range is 0~7. The larger the value, the higher the degree of trajectory fusion. When not selected, it defaults to PL=0.
CR, blend radius, unit: mm. The parameter PL is deprecated.
FPT: Determine the end point of an arc

Parameters in advanced mode: [joint/pose]: Joint angle or pose variable. When there is this optional parameter, it means to use the specified P or V variable, if not, it means to use the taught fixed point. P or V variable allows manual modifications;
[ACC=int]: Acceleration percentage, the range is 0~100;
[DEC=int]: Deceleration percentage, the range is 0~100;
[TOOL#(int]: optional parameter, tool coordinate system number, range: 0~7;
[UNTIL bool]: When the condition is established, this point does not run, if not, it runs this point

Example:

```
MOVC V=500 MM/S CR=50.0MM
MOVC V=500 MM/S CR=50.0MM
MOVC V=500 MM/S CR=50.0MM
//Three fixed points taught by circular interpolation, speed V
   =500MM/S, and blending radius of 50mm,
MOVC P000 AV=100MM/S CR=0.0MM
MOVC P001 AV=100MM/S CR=0.0MM
MOVC P002 AV=100MM/S CR=0.0MM
// Move to P000, P001, P002 in turn with circular
   interpolation and speed AV=100MM/S
SET I000 100
MOVC P000 AV=I000 CR=100.0MM
MOVC P001 AV=I000 CR=100.0MM
MOVC P002 AV=I000 CR=100.0MM
//Move to P000, P001, P002 points in turn with circular
   interpolation, speed AV=I000 MM/S, and blend radius of 100
   mm
```

2.4.4 Drag path re-perform command (MOV)

```
MOVDRAG VJ=int DRAG=int JOB: dragfile
```

Function: Drag path re-perform command. The robot will re-perform the motion trajectory in the specified drag file.

Parameters: VJ=int: the speed of the movement from current position to the starting point of the drag trajectory in the manner of joint interpolation, range 1%~100%
DRAGV=int: Max. joint speed of the trajectory re-perform movement, range 1% ~100% (The combined 6 joints will cause constrains on max. joint speed.)
JOB: dragfile: dragged track file, which is a special jbi file that cannot be opened and edited directly on the teach pendant.

Example:

```
MOVDRAG VJ=60% DRAGV=25% JOB: run
// From current position move to the start point of the
   trajectory of run.jbi with speed of 60%. And re-
   perform the trajectory of run.jbi with speed of 25%.
```

2.4.5 Micro-segment interpolation instruction (MOV)

```
MOVML VJ = int type#[addr] JOB: filename
```

Function: Micro interpolation instructions, used to reproduce the point file

Parameters: VJ =int: joint motion speed, the range is 1%~100%;
type#[addr]: wait for the signal to be valid after the joint moves to the starting point of the offline file
type: selectable signal type, IN digital input or M virtual input;
addr: address, int type
JOB: filename: offline file name, only JBI files whose header is dragfile, trajfile, pathfile or loadfile can be recognized.

Example:

```
NOP
LOADML JOB:dragfile1
MOVML VJ=100% IN#(4) JOB:dragfile1
UNLOADML JOB:dragfile1
END
//Offline file format:
///pathfile //File header, indicating to the robot that this
   is a path file
version2.15.x //2.15.0 and above versions support new formats
```

```
    (add IO, coordinate system)
interval [2]ms //Sampling time interval, supports integers
   greater than 1ms
//The joint angle of the first point or the inverse solution
   reference value of the first pose, unit: degree
refJointPos [-119.95,-167.84,-55.02,-175.47,87.41,-24.31]
length [3729] //Length of the following data (number of data
   rows)
postype [joint] //Set data type to joint angle or pose joint/pose
outputNumber[23,25] #The serial number of the output IO,
   supports up to 6 definitions, and at least one IO needs to
    be written. If it is not enabled, the output command is
   not used.
refFrame [0,0,0,0,0,0] #The coordinate system on which the
   data is based. If the base coordinate system is used, it
   must be all 0.
[-119.952,-167.839,-55.0197,-175.474,87.4069,-24.3066]
[-119.902,-167.789,-55.0697,-175.424,87.4569,-24.3566]
output[0,1] #Corresponding to the IO state that sets the
   outputNumber to define the serial number, up to 6 are
   supported. If the state is not set, the above IO is not
   used.
[-119.852,-167.739,-55.1197,-175.374,87.5069,-24.4066]
[-119.802,-167.689,-55.1697,-175.324,87.5569,-24.4066]
....
[-120,-170,-50.2404,-183.597,74.8063,-38.711]
[-120,-170,-50.3607,-183.597,74.8063,-38.6572]
```

2.4.6 Joint motion command (MOVE)

```
MOVEJ V/VJ=int [CR=int] [A/ACC=int] [D/DEC=int] P/LP/ConstP [UNTIL bool]
```

Function: Joint motion command. The robot will move by joint interpolation when executing this command.

Parameters: Only one of the following waypoints P/LP/ConstP (standard parameters) can be selected: P/LP variable, the range is P000-P255/LP000-LP255, ConstP: joint angle, double pos[6], the range is [-360,360], in the unit of angle, P/ConstP variable allows manual modifications; V/VJ: speed parameter. The value of speed can be directly entered or set by using VJ variable. V represents the real value. The default unit is DEG/S, and the range is [1, 260], and VJ represents percentage, and the range is [1,

100]; CR: blend radius, optional parameter, range is [0, 2608], unit: mm, if not transmitted, then the value is equal 0;
[A/ACC=int] acceleration, you can choose to enter A/ACC or choose not to enter, if A/ACC=0 or not enter, then run at 80% of the default acceleration:
A: absolute acceleration, default unit is MM/S^2, range is [0, 1500];
ACC: percentage of acceleration, default unit: %, range: [1, 100];
[D/DEC=int] deceleration, you can choose to enter D/DEC, or choose not to enter, if D/DEC=0 or not enter, then run at 80% of the default deceleration;
D: absolute deceleration, default unit is MM/S^2, range is [0, 1500];
DEC: percentage of deceleration, default unit %, range [1, 100];
[UNTIL]: optional parameter. The condition for the execution of this motion instruction, to judge a variable or constant. When the conditions are met, this point does not run, otherwise, it runs this point. Multiple conditions can be superimposed in series by logical operators 1{or} and &{and}. Users can select the parameter OUT, which is used to record the current joint angle or pose in the variable P/V when the UNTIL condition is fulfilled.

Example:

```
MOVEJ VJ=50 CR=50.0MM P000
//Move to the P000 point by joint interpolation,50% speed and
   blend radius of 50mm
MOVEJ VJ=50% CR=0.0MM ConstP=[-72.389,-82.861,127.856,-138.839,
      89.999,-0.000]
//Move to the ConstP point by joint interpolation and 50% speed
MOVEJ VJ=50 CR=50.0MM ACC=50 DEC=50 P000 UNTIL M#(528)=ON
//Move to the P000 point by 50% speed, 50% acceleration, 50%
   deceleration and blend radius of 50mm, until M#(528) is 1, then
   the movement stops
SET LP000 P001
MOVEJ V=50 CR=0.0MM A=100 D=10 LP000
//Move to the LP000 point by 50°/s² joint speed, 100°/s² acceleration,
   10°/s² deceleration, and blend radium of 0mm
```

Note: The actual speed, acceleration, and deceleration of motion depend on the kinematic parameter limits. To adjust the parameters, go to Parameter Settings > Kinematic Parameters to modify the range.

2.4.7 Linear motion command (MOVE)

```
MOVEL V'=int CR=int FPT [A/ACC=int] [D/DEC=int] [TOOL#(int)/FLANGE]
      [USER#(int)] [REF=P/ConstP] P/LP/V/LV/ConstV/ConstP [UNTIL bool]
```

Function: Linear motion command. The robot will move by linear interpolation when executing this command.

Parameters: Only one of the following waypoints P/LP/V/LV/ConstV/ConstP (standard parameters) can be selected: P/LP variable, the range is P000-P255/LP000-LP255, V/LV variable, the range is V000-V255/LV000-LV255,ConstP joint angle, double pos[6], the range is [-360,360], in the unit of angle; ConstV terminal pose, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-1e+09~1e+09], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π,π], P/V/ConstV/ConstP variable allows manual modifications;

The following speed parameters can be set to one of V, VR, AV, AVR: V= linear speed, range is 0.01-3000 MM/S; VR= rotation angular speed, range is 1-300°/S; AV= absolute linear speed, range is 0.01-3000mm /S; AVR= absolute rotation angular speed, range is 1-300°/S;

CR: blend radius, optional parameter, range is [0, 2608], unit: mm, if not transmitted, then the value is equal 0;

[A/ACC=int] acceleration, you can choose to enter A/ACC or choose not to enter, if A/ACC=0 or not enter, then run at 80% of the default acceleration:

A: absolute acceleration, default unit is MM/S^2, range is [0, 1500];

ACC: percentage of acceleration, default unit: %, range: [1, 100];

[D/DEC=int] deceleration, you can choose to enter D/DEC, or choose not to enter, if D/DEC=0 or not enter, then run at 80% of the default deceleration;

D: absolute deceleration, default unit is MM/S^2, range is [0, 1500];

DEC: percentage of deceleration, default unit %, range [1, 100];

[TOOL#(N)/FLANGE] : tool coordinate system, optional; if it is not entered, the current tool is used;

[TOOL#(N)] : tool coordinate system, N is the tool number, range is [0, 7];

[FLANGE] : flange;

[USER#(N)] : user coordinate system, optional. N is the user number and the range is [0, 7]. If it is not entered, the base coordinate system is used, and if it is entered, the specified user coordinate system is used;

[UNTIL]: optional parameter. The condition for the execution of this motion instruction, to judge a variable or constant. When the conditions are met, this point does not run, otherwise, it runs this point. Multiple conditions can be superimposed in series by logical operators 1{or} and &{and}. Users can select the parameter OUT which is used to record the current joint angle or pose in the variable P/V when the UNTIL condition is fulfilled.

[REF]: optional parameter, if not transmitted, the current position will be taken as the reference joint point. Reference solution required for inverse solution, in two expansions:

P variable assignment, the range is [P000-P255], the unit is angle;

Defined variable assignment, REF = [j1,j2,j3,j4,j5,j6], the range is [-360,360], and

the unit is angle.

Note:

1. When the speed input by the user in the instruction interface exceeds the parameter range and the value is less than the minimum value, the robot will run with the minimum value of the parameter range as the actual speed, when the speed input by the user in the instruction interface exceeds the parameter range and the value is greater than the maximum value, the robot will run with the maximum value of the parameter range as the actual speed. To adjust the parameters, go to Parameter Settings -> Kinematic Parameters to modify the range.
2. When the information about MOVEL waypoints are all of the joint type (ConstP, P), the position of the target joint angles will not be affected by passing or not passing the tool and user parameters.

Example:

```
MOVEL VJ=100 CR=50.0MM P000
//Move to the P000 point by linear interpolation, linear speed of
   100mm/s and blend radius of 50mm
MOVEL VJ=100 CR=50.0MM ConstP=[-72.389,-82.861,127.856,
      -138.839,89.999,-0.000]
//Move to the ConstP point by linear interpolation and 50%
rotation angular velocity of 100°/s and blend radius of 50mm
MOVEL V=100 ACC=50 DEC=50 TOOL#(0) V000 REF=[-72.389,-82.861,
      127.856,-138.839,89.999,-0.000] UNTIL M#(528)=ON
//Under tool 0, take ConstP as the reference point of inverse
   solution, and move to the V000 point by linear speed of 100mm/s,
   50% acceleration and 50% deceleration until M# (528) is 1, then
   the movement stops
SET LV000 V001
MOVEL V=50 CR=50.0MM A=100 D=10 FLANGE LV000
//Under the flange coordinate system, move to LV000 point by 50°/s
   speed, 100°/s2 acceleration, 10°/s2 deceleration, and the blend
   radius of 50mm
MOVEL V=50 CR=0.0MM A=100 D=10 USER#(0) ConstV=[399.144,210.115,
      389.458,3.0932753,0.0003718,-1.3607399] REF=P000
//Under the No.0 user coordinate system, take P000 as the reference
   point of inverse solution. Move to the ConstV point by 50°/s
   speed, 100°/s2 acceleration and the blend radius of 0mm
```

2.4.8 Circular motion command (MOVE)

```
MOVEC V'=int [CR=int] [A/ACC=int] [D/DEC=int] [TOOL#(int)/FLANGE] [USER#(int)]
      P/LP/V/LV ConstP/ConstV P/LP/V/LV/ConstP/ConstV [REF] [UNTIL]
```

Function: Circular motion command. When the robot executes this command, it will complete a segment of circular path movement through the three points taught by circular interpolation.

Parameters: Only one of the intermediate route waypoints P/LP/V/LV/ConstV/ConstP (standard parameters) can be selected: P/LP variable, the range is P000-P255/LP000-LP255, V/LV variable, the range is V000-V255/LV000-LV255, ConstP joint angle, double pos[6], the range is [-360,360], in the unit of angle, ConstV terminal pose, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-1e+09~1e +09], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π,π], P/V/ ConstV/ConstP allows manual modifications; The following speed parameters can be set to one of V, VR, AV, AVR: V= linear speed, range is 0.01-3000 MM/S; VR= rotation angular speed, range is 1-300°/S; AV= absolute linear speed, range is 0.01-3000mm /S; AVR= absolute rotation angular speed, range is 1-300°/S;
CR: blend radius, optional parameter, range is [0, 2608], unit: mm, if not transmitted, then the value is equal 0;
[A/ACC=int] acceleration, you can choose to enter A/ACC or choose not to enter, if A/ACC=0 or not enter, then run at 80% of the default acceleration:
A: absolute acceleration, default unit is MM/S^2, range is [0, 1500];
ACC: percentage of acceleration, default unit: %, range: [1, 100];
[D/DEC=int] deceleration, you can choose to enter D/DEC, or choose not to enter, if D/DEC=0 or not enter, then run at 80% of the default deceleration;
D: absolute deceleration, default unit is MM/S^2, range is [0, 1500];
DEC: percentage of deceleration, default unit %, range [1, 100];
[TOOL#(N)/FLANGE] : tool coordinate system, optional; if it is not entered, the current tool is used;
[TOOL#(N)] : tool coordinate system, N is the tool number, range is [0, 7];
[FLANGE] : flange;
[USER#(N)] : user coordinate system, optional. N is the user number and the range is [0, 7]. If it is not entered, the base coordinate system is used, and if it is entered, the specified user coordinate system is used;
[UNTIL]: optional parameter. The condition for the execution of this motion instruction, to judge a variable or constant. When the conditions are met, this point does not run, otherwise, it runs this point. Multiple conditions can be superimposed in series by logical operators 1{or} and &{and}. Users can select the parameter OUT which is used to record the current joint angle or pose in the variable P/V when the UNTIL condition is fulfilled.

[REF]: optional parameter, if not transmitted, the current position will be taken as the reference joint point. Reference solution required for inverse solution, in two expansions:
P variable assignment, the range is [P000-P255], the unit is angle;
Defined variable assignment, REF = [j1,j2,j3,j4,j5,j6], the range is [-360,360], and the unit is angle.
Only one of the target waypoints P/LP/V/LV/ConstV/ConstP (standard parameters) can be selected: P/LP variable, the range is P000-P255/LP000-LP255, V/LV variable, the range is V000-V255/LV000-LV255, ConstP joint angle, double pos[6], the range is [-360,360], in the unit of angle, ConstV terminal pose, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-1e+09~1e +09], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π,π], P/V/ ConstV/ConstP variable allows manual modifications; The following speed parameters can be set to one of V, VR, AV, AVR: V= linear speed, range is 0.01-3000 MM/S; VR= rotation angular speed, range is
1-300°/S; AV= absolute linear speed, range is 0.01-3000mm /S; AVR= absolute rotation angular speed, range is 1-300°/S;
CR: blend radius, optional parameter, range is [0, 2608], unit: mm, if not transmitted, then the value is equal 0;
[A/ACC=int] acceleration, you can choose to enter A/ACC or choose not to enter, if A/ACC=0 or not enter, then run at 80% of the default acceleration:
A: absolute acceleration, default unit is MM/S^2, range is [0, 1500];
ACC: percentage of acceleration, default unit: %, range: [1, 100];
[D/DEC=int] deceleration, you can choose to enter D/DEC, or choose not to enter, if D/DEC=0 or not enter, then run at 80% of the default deceleration;
D: absolute deceleration, default unit is MM/S^2, range is [0, 1500];
DEC: percentage of deceleration, default unit %, range [1, 100];
[TOOL#(N)/FLANGE] : tool coordinate system, optional; if it is not entered, the current tool is used;
[TOOL#(N)] : tool coordinate system, N is the tool number, range is [0, 7];
[FLANGE] : flange;
[USER#(N)] : user coordinate system, optional. N is the user number and the range is [0, 7]. If it is not entered, the base coordinate system is used, and if it is entered, the specified user coordinate system is used;
[UNTIL]: optional parameter. The condition for the execution of this motion instruction, to judge a variable or constant. When the conditions are met, this point does not run, otherwise, it runs this point. Multiple conditions can be superimposed in series by logical operators 1{or} and &{and}. Users can select the parameter OUT which is used to record the current joint angle or pose in the variable P/V when the UNTIL condition is fulfilled.

[REF]: optional parameter, if not transmitted, the current position will be taken as the reference joint point. Reference solution required for inverse solution, in two expansions: P variable assignment, the range is [P000-P255], the unit is angle; Defined variable assignment, REF = [j1,j2,j3,j4,j5,j6], the range is [-360,360], and the unit is angle.

Note:

1. When the speed input by the user in the instruction interface exceeds the parameter range and the value is less than the minimum value, the robot will run with the minimum value of the parameter range as the actual speed, when the speed input by the user in the instruction interface exceeds the parameter range and the value is greater than the maximum value, the robot will run with the maximum value of the parameter range as the actual speed. To adjust the parameters, go to Parameter Settings > Kinematic Parameters to modify the range.
2. When the information about MOVEC waypoints are of the joint type (ConstP, P), the position of the target joint angles will not be affected by passing or not passing the tool and user parameters.

Example:

```
MOVEC V=100 CR=50.0MM P000 P001
//Move to the P001 point by circular interpolation, P000 as the
   center point, 100mm/s linear speed and blend radius of 50mm
MOVEC VR=100 CR=50.0MM V000 ConstP=[-72.389,-82.861,127.856,-138.839,
      89.999,0.000]
//Move to the ConstP point by circular interpolation, V000 as the
   center point, 100°/s rotation angular velocity, and blend radius
   of 50mm
MOVEC V000 VR=100 CR=50.0MM TOOL#(0) USER#(0) ConstV=[399.144,
      210.115,389.458,3.0932753,0.0003718,-1.3607399]
//Under tool 0 and user 0 coordinate system, move to the V000 point
   by ConstV as the center point, 100°/s rotation angular velocity
   and blend radius of 50mm
MOVEC V=100 FLANGE ConstP=[-72.389,-82.861,127.856,-138.839,89.999,
      0.000] ConstV=[399.144,210.115,389.458,3.0932753,0.0003718,
      -1.3607399]
//Under the flange coordinate system, take ConstP as the center point
   and move to the ConstV point by a linear speed of 100mm/s
   radius of 50mm
MOVEC V=100 CR=0.0mm ACC=50 DEC=50 V000 V001 REF=[-72.389,-82.861,
      127.856,-138.839,89.999,-0.000] UNTIL M#(528)=ON
//Take V000 as the center point, ConstP as the reference point for
   inverse solution, move to the V001 point by linear velocity of
   100mm/s, 50% acceleration, 50% deceleration, and the blend radius
   is 0mm, until M#(528) is 1, then the movement stops
```

2.4.9 Drag path re-perform command (MOVE)

```
MOVEDRAG VJ=int DRAGV=int JOB: filename
```

Function: Drag path re-perform command. The robot will re-perform the motion trajectory in the specified drag file.

Parameters: VJ=int: the speed of the movement from current position to the starting point of the drag trajectory in the manner of joint interpolation, range 1%~100%
DRAGV=int: the speed of the trajectory re-perform movement, range 1% ~100%. When the value is equal to 100%, the max. speed of the trajectory re-perform movement is the max. joint speed of the robot (Note: The combined 6 joints will cause constrains on max. joint speed.). When the parameter DRAGV is not entered, the max speed of the trajectory re-perform movement is the original speed pf the trajectory movement.
JOB: dragfile: dragged track file, which is a special jbi file that cannot be opened and edited directly on the teach pendant.

Example:

```
MOVEDRAG VJ=1% DRAGV=1% JOB: path
// From current position move to the start point of the
   trajectory of path.jbi with speed of 1%. And re-
   perform the trajectory of path.jbi with speed of 1%.
```

2.4.10 Micro-segment interpolation instruction (MOVE)

```
MOVEML VJ = int type#[ADDR] JOB: filename
```

Function: Micro interpolation instructions, used to reproduce the point file

Parameters: VJ =int: joint motion speed, the range is 1%~100%;
type#[addr]: wait for the signal to be valid after the joint moves to the starting point of the offline file
type: selectable signal type, IN digital input or M virtual input;
addr: address, int type
JOB: filename: offline file name, only JBI files whose header is dragfile, trajfile, pathfile or loadfile can be recognized.

Example:

```
NOP
LOADML JOB:dragfile1
MOVEML VJ=100% IN#(4) JOB:dragfile1
UNLOADML JOB:dragfile1
END
//Offline file format:
///pathfile //File header, indicating to the robot that this
   is a path file
version2.15.x //2.15.0 and above versions support new formats
    (add IO, coordinate system)
interval [2]ms //Sampling time interval, supports integers
   greater than 1ms
//The joint angle of the first point or the inverse solution
   reference value of the first pose, unit: degree
refJointPos [-119.95,-167.84,-55.02,-175.47,87.41,-24.31]
length [3729] //Length of the following data (number of data
   rows)
postype [joint] //Data type joint angle or pose joint/pose
outputNumber[23,25] #The serial number of the output IO,
   supports up to 6 definitions, and at least one IO needs to
    be written. If it is not enabled, the output command is
   not used.
refFrame [0,0,0,0,0,0] #The coordinate system on which the
   data is based. If the base coordinate system is used, it
   must be all 0.
[-119.952,-167.839,-55.0197,-175.474,87.4069,-24.3066]
[-119.902,-167.789,-55.0697,-175.424,87.4569,-24.3566]
output[0,1] #Corresponding to the IO state that sets the
   outputNumber to define the serial number, up to 6 are
   supported. If the state is not set, the above IO is not
   used.
[-119.852,-167.739,-55.1197,-175.374,87.5069,-24.4066]
[-119.802,-167.689,-55.1697,-175.324,87.5569,-24.4066]
....
[-120,-170,-50.2404,-183.597,74.8063,-38.711]
[-120,-170,-50.3607,-183.597,74.8063,-38.6572]
```

2.4.11 Initialize transparent transformation process

```
TTINIT T=double  LOOKAHEAD=double SMOOTHNESS=double
```

Function: Initialize transparent transformation process

Parameters: T=double: Sampling time. Robot takes positon information once in this time interval. Range: 0.002~0.1. Unit: second.
LOOHAHEAD=double: Forward time. Robot wait for this time before start taking position in queue. Range: 0.01~1. Unit: second.
SMOOTHNESS=double: Trajectory gain. It will be used to smooth the path. Range: 0~1.
NOTE: LOOKAHEAD time must be longer than the sampling time.

Example:

```
TTINIT T=0.004S LOOKAHEAD=0.1S SMOOTHNESS=0.2
// The sampling time is 0.004s, the look-ahead time is
   0.1s, and the trajectory smoothness is 0.2
```

2.4.12 Transparent transformation start position

```
TTSTARTJOINT  joint
```

Function: Transparent transformation start position. Default as current position.

Parameters: joint: joint position variable P

Example:

```
TTSTARTJOINT P001
//Save the current position to P001 and use it as the
   start position of current transparent transformation
   process
```

2.4.13 Transparent transformation end

```
TTSTOP
```

Function: Transparent transformation ends.

Parameters: None

Example:

```
TTSTOP
```

2.4.14 Load job file

```
LOADML JOB:filename
```

Function: The content of the executed file is preloaded into the memory, which can reduce the time when using the MOVEML instruction to run the file

Parameters: JOB: filename: offline file name, only JBI files whose header is dragfile, trajfile, pathfile or loadfile can be recognized.

Example:
```
LOADML JOB: path_path
```

Note: After the system restarts, the files in the memory will be deleted. A maximum of 32 files can be stored in the memory, and the first file will be replaced when the 33rd file is run, and so on.
This command is applicable to v2.11.0 and above.

2.4.15 Unload job file

```
UNLOADML JOB:filename
```

Function: Delete the executable file in the memory

Parameters: JOB: filename: offline file name, only JBI files whose header is dragfile, trajfile, pathfile or loadfile can be recognized.

Example:
```
UNLOADML JOB:path_path
```

Note: This command is applicable to v2.11.0 and above.

2.4.16 Transparent transmission to increase target location

```
TTTARGETJOINT joint or TTTARGETJOINT pose
```

Function: Transparent transmission increases the target location

Parameters: joint: joint position, P variable.
pose: spatial pose, V variable

Example:
```
//Increase the target point P001

TTTARGETJOINT P001

//Increase target point V001

TTTARGETJOINT V001
```

2.5 Mathmatic Instructions

For the parameters TOOL, USER and UNTIL, the subscript values are of type int and have the range. In the phase of decoding, the internal optimization will be done in the system. That is, when assigning a fractional or float variable to an integer variable, only the integer part (not rounded, but the fractional part is completely discarded) is used.

2.5.1 Increment instruction

```
INC variable
```

Function: Variable value incremental (plus 1) instructions

Parameters: variable: object variables, support one of B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], (see Section 2.1 for details).

Example:

```
INC LB000
//increase the value of LB000 by 1
```

2.5.2 Decrement instruction

```
DEC variable
```

Function: Decreasing variable value (minus 1) instructions

Parameters: variable: object variables, support one of B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], (see Section 2.1 for details).

Example:

```
DEC LI002
//decrease the value of LI002 by 1
DEC B[B5]
//If B005=7, DEC B[B5] is equivalent to DEC B007
```

2.5.3 Addition instruction

```
ADD variable#1 variable#2
```

Function: Addition instruction, variable #1 and variable #2 will be added and the result will be stored in variable #1.

Parameters: variable#1：Variable#1,can be one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
variable#2: Variable#2, can be an integer, float, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
Note: When doing this with integer and floating-point variables, if the result is stored in an integer variable, only the integer part of the result is stored (no rounding, but the decimal part is discarded completely).

Example:

```
ADD B001 5
//Add 5 to variables B001 and the result is stored in
   B001
ADD D001 I002
//Add the variable D001 and variables I002 and the result
    is stored in D001
ADD P001(2) 25
//Add 25° to the 3rd asix angle of P001
```

2.5.4 Subtraction instruction

```
SUB variable#1 variable#2
```

Function: Subtraction instruction, variable #1 minus variable #2, the result is stored in variable #1.

Parameters: variable#1: Variable#1, can be one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
variable#2: variable#2, can be integer, float, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details.
Note: When doing this with integer and float variables, if the result is stored in an integer variable, only the integer part of the result is stored (no rounding, but the decimal part is discarded completely).

Example:

```
SUB B001 5.8
//subtract 5.8 from variable B001 and the result is
   stored in B001
SUB I003 LD004
//subtract the value of variable LD004 from that of I003
SUB P006(2) 90
//Subtract 90° from the value of joint_3 of variable P006
```

2.5.5 Multiplication instruction

```
MUL variable#1 variable#2
```

Function: Multiplication instruction, variable #1 is multiplied by variable #2 and the result of the is stored in variable #1.

Parameters: variable#1: Variable#1, can be one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
variable#2: Variable#2, can be integer, float, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
Note: When doing this with integer and floating-point variables, if the result is stored in an integer variable, only the integer part of the result is stored (no rounding, but the decimal part is discarded completely).

Example:

```
MUL B001 I002
//variable B001 values and variables I002 and the result
   is stored in the variable B001 Medium
```

2.5.6 Division instruction

```
DIV variable#1 variable#2
```

Function: Division instruction, divide #1 by #2 and store the result in variable #1.

Parameters: variable#1: Variable#1, can be one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
variable#2: Variable#2, can be integer, float, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details.
Note: When doing this with integer and floating-point variables, if the result is stored in an integer variable, only the integer part of the result is stored (no rounding, but the decimal part is discarded completely).

Example:

```
DIV B001 50.1
//If B001=100, B001 divide by 50.1 and take only the
   integer part of the result to be stored in B001. B001
   =1 in this case
```

Note: When the divisor in the division operation instruction is set to 0, the robot reports a division by zero error, and the divided variable still becomes 0

2.5.7 Residual calculation

```
MOD variable#1 variable#2
```

Function: Residual calculation, divide variable #1 by variable #2 and store the residual of the calculation into variable #1.

Parameters: variable#1: Variable#1, can be one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
variable#2: Variable#2, can be integer, float, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
Note: This operation does not accept fractions or floats. When these two types of variables or constants are involved, only integers (not rounded) are used. The system does not report errors.

Example:

```
MOD B001 50.98
//if B001=51, after MOD calculation round-up the result
   to the nearest integer and stored it to B001. B001=1
   in this case
```

2.5.8 Bitwise and operation instruction

```
AND variable#1 variable#2
```

Function: Bitwise and operation instruction. It will convert the values of variable #1 and variable #2 to binary format and then perform AND logic operation.
Then convert the result to decimal integer and store it in variable #1.

Parameters: variable#1: Variable#1, one of B, I, LB, LI, B[*], LB[*], I[*], or LI[*] can be selected; (see Section 2.1 for details).
variable#2: variable#2, can be one of integer, float, B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details .
Note: If the value of the variable is negative or float, the system will take the absolute value of the integer part of the variable, convert it to binary and bring it into the operation.

Example:

```
AND B001 B002
// Bitwise AND operation between B001 and B002. The
   result is stored in the variable B001
```

2.5.9 Bitwise or operation instruction

```
OR variable#1 variable#2
```

Function: Bitwise or operation instruction.
Take the values of variables #1 and #2 and convert them to binary format, then perform OR logic operation and convert the result to a decimal integer to be stored in variable #1.

Parameters: variable#1 : Variable#1, can choose one of B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details).
variable#2: variable#2,can be integer, float, B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details.
Note: If the value of the variable is negative or decimal, the system will take the absolute value of the integer part of the variable, convert it to binary and bring it into the operation.

Example:

```
OR B001 B002
// Bitwise OR operation between B001 and B002. The result
    is stored in the variable B001
```

2.5.10 Opposition operation instruction

```
NOT variable#1 variable#2
```

Function: Opposition operation instruction.
Converts the value of variable #2 to binary and then performs a NOT logic operation to convert the result to a decimal integer and stored in variable #1.

Parameters: variable#1：Variable#1, can choose one of B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details).
variable#2：Variable#2, can choose one of integer, float, B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details).
Note: If the value of the variable is negative or decimal, the system will take the absolute value of the integer part of the variable, convert it to binary and bring it into the operation.

Example:

```
NOT B001 B002
//After taking the non-logic value of B001 and B002, the
   result is stored in variable B001. If the value of
   B002 is less than or equal to 255, the value of B002
   is converted into an eight-digit binary number and
   then the inverse operation is performed, and converted
    into a decimal number and assigned to B001.
//If the value of B002 is greater than 255, it will be
   converted into a multi-digit binary number (more than
   eight digits) for inversion calculation, and converted
    into a decimal number and assigned to B001.
```

2.5.11 Different or operation instruction

```
XOR variable#1 variable#2
```

Function: Different or operation instruction.
Take the values of variables #1 and #2 and convert them to binary, then perform an XOR logical operation and convert the result to a decimal integer to be stored in variable #1.

Parameters: variable#1: Variable#1,one of B, I, LB, LI, B[*], LB[*], I[*], or LI[*] can be selected; (see Section 2.1 for details).
variable#2: variable#2, can be integer, float, B, I, LB, LI, B[*], LB[*], I[*], or LI[*]; (see Section 2.1 for details).
Note: If the value of the variable is negative or decimal, the system will take the absolute value of the integer part of the variable, convert it to binary and bring it into the operation.

Example:

```
XOR B001 B002
// Bitwise XOR operation between B001 and B002. The
   result is stored in the variable B001
```

2.5.12 Calculate the distance between two points.

```
DIST D#(int) joint#1 joint#2
```

Function: Calculate the distance between two giving points.

Parameters: D#(): Variable to save the result, can be of type D, LD, D[*] or LD[*], (see Section 2.1 for details).
joint#1, joint#2: Position variables to be calculated, either of type P, LP, P[*] or LP[*], (see Section 2.1 for details).

Example:

```
DIST D000 P000 LP001
//Calculate the distance between P001 and LP001and store
   the result in D000
```

2.5.13 Specify the coordinate system command

```
CCOOD frame
```

Function: Specify the coordinate system command. When this instruction is used in a JBI script program, the calculation after this instruction line is carried out in the specified coordinate system.

Parameters: type of coordinate system.
CART = Cartesian coordinates, JOINT = joint coordinates, USER#()=user coordinate system (range 0-7); TOOL#()=tool coordinate system number (range 0-7)

Example:

```
// set the 3rd axis angle of P001 to 60.00 degree
CCOOD CART
// designated as a Cartesian coordinate system.
SET P001(2) 100.00
// set the offset distance in Z direction (3rd element in
    P001 in Cartesian frame) to 100.00mm
```

Note: The actual position of the taught fixed point does not change in different coordinate systems. This command only affects the calculation and assignment of related variables in the content from the specified coordinate system command line to the next specified coordinate system command line.
The specified tool coordinate system is applicable to 2.13.0 and above.

2.5.14 Pose Multiplication

```
POSEMUL pose#1 pose#2
```

Function: Pose multiplication. Pose1 as the center of rotation, pose2 as the offset pose matrix, and pose1 is offset.

Parameters: pose#1: Cartesian pose variable V, size: 0~255
pose#2: Cartesian pose variable V, size: 0~255

Example:

```
POSEMUL V001 V002
// According to the offset defined in V002, append the
   transformation to V001
```

2.5.15 Inverse Kinematics

```
POSETOJOINT pose#1 joint#1
```

Function: Convert Cartesian pose to joint position

Parameters: pose#1: Cartesian pose variable V, size: 0~255. The spatial variable to be converted
joint#1: Joint position variable P, size: 0~255. Variables for saving joint coordinates derived from the inverse solution. If this variable is not null, it will also be used as the reference joint for the inverse calculation. Note that the reference point for the inverse calculation is the value saved by the current P variable.

Example:
```
POSETOJOINT V001 P001
// Covert V001 joint position and store the result in
   variable P001
```

2.5.16 Forward Kinematics

```
JOINTTOPOSE joint#1 pose#1
```

Function: Convert joint position to Cartesian pose.

Parameters: joint#1: joint position variable P, size 0~255. Joint position variable to be converted
pose#1: Cartesian pose variable V, size 0~255. Variables used to store Cartesian pose calculated from the positive solution

Example:
```
JOINTTOPOSE P005 V005
// Covert P005 to Cartesian pose and store the result in
   variable V005
```

2.5.17 Pose inversion

```
POSEINV pose
```

Function: Perform the inversion calculation of the pose, and then assign the value to V1 after the inversion of V1.

Parameters: pose: spatial position variable V range 0~255

Example:
```
POSEINV V1
```

Note: This command is applicable to v2.11.0 and above.

2.5.18 Calculate the pose variation

```
POSESUB pose#1 pose#2
```

Function: Calculate the pose variation. Pose#1 subtracts pose#2. Save the calculations into pose#1.

Parameters: pose#1: spatial position variable V, range 0~255. It is used to save the calculations of the pose variation.
pose#2: spatial position variable V, range 0~255.

Example:

```
NOP
POSESUB V000 V001
END
```

2.6 Set and Get Instructions

For the parameters TOOL, USER and UNTIL, the subscript values are of type int and have the range. In the phase of decoding, the internal optimization will be done in the system. That is, when assigning a fractional or float variable to an integer variable, only the integer part (not rounded, but the fractional part is completely discarded) is used.

2.6.1 Variable assignment instruction

```
SET variable#1 variable#2
```

Function: variable assignment instruction, assign the value of variable #2 to variable #1 and save the new variable #1.

Parameters: variable#1: Variable#1, can be one of B, I, D, P, LB, LI, LD, LP; (see Section 2.1 for details).
variable#2: variable#2, can be integer, float, B, I, D, P, LB, LI, LD, LP; (see Section 2.1 for details).
Note: When assigning a fractional or float variable to an integer variable, only the integer part (not rounded, but the fractional part is completely discarded) is used.
In addition, P variables are not supported to be assigned as arrays, but must be subdivided into single elements.

Example:

```
SET D001 8.88
//Set D001 value to 8.88
SET B001 6
//Set B001 value to 6
SET B003 D001
//Round up D001 to the nearest integer and assign it to
   B003
SET P020(4) 100
//Set the 4th axis angle of P020 to 100°
```

2.6.2 Joint position assignment instruction

```
SETJOINT P001 ConstJoint
SETJOINT P001 ConstP=[double, double, double, double, double, double]
```

Function: Joint position assignment instruction.
Assign the angular values of the current 6 joints of the robot to a designated P variable

Parameters: joint: joint position variable P, size 0 to 255.
ConstJoint: double, double, double, double, double, double angular values J1,J2,J3,J4,J5,J6, composed of 6 axes, can be modified manually by clicking "point 1" in the edit mode, generated auto-matically based on the current robot position.
ConstP: ConstP=[double, double, double, double, double, double] angular values J1,J2,J3,J4,J5,J6, composed of 6 axes, can be modified manually by clicking "point 1" in the edit mode, generated auto-matically based on the current robot position.
Note: ConstJoint/ConstP are mutually exclusive. ConstJoint is compatible with the old SETJOINT command, which makes it can edit the data of 6 axes.

Example:

```
SETJOINT P002 1.111,2.222,3.333,4.4444,5.5555,6.6666
SETJOINT P002 ConstP=[1.111,2.222,3.333,4.444,5.555,6.666]
//Assign the current robot position shown in 6 angular
   numbers to P002 variable. You can view it under tab "
   Monitor"
//Note: Elements in P variable are rounded to the nearest
    3 decimal places.
```

Note: It must be inserted in the servo on state; it must be in the P variable open state when set, and the joint assignment will report errors when it is not open.

2.6.3 Spatial position assignment instruction

```
SETPOSE V001 ConstPose
SETPOSE V001 ConstV=[double, double, double, double, double, double]
```

Function: Spatial position assignment instruction.
Assign the dimensional values of robot's current Cartesian pose to a designated V variable.

Parameters: pose: spatial position variable V, size 0 to 255.
ConstPose: double, double, double, double, double: In the base coordinate system, the spatial co-ordinate values x, y,z, rx, ry, rz, consist of transformation and rotation values, unit of rx, ry ad rz is rad. Can be modified manually by clicking "point 1" in the edit mode, automatically generated based on the current robot position.
ConstV: ConstV=[double, double, double, double, double, double]: In the base coordinate system, the spatial co-ordinate values x, y, z, rx, ry, rz, consist of transformation and rotation values, unit of rx, ry and rz is rad. Can be modified manually by clicking "point 1" in the edit mode, automatically generated based on the current robot position. Note: ConstPose/ConstV are mutually exclusive. ConstPose is compatible with the old SETPOSE command, which makes it can edit the data of 6 axes.

Example:

```
SETPOSE V002 10.0000000, 20.0000000, 30.0000000, 2.9478373,
0.9326294, -2.0134086
SETPOSE V002 ConstV=[10.0000000, 20.0000000, 30.0000000,
2.9478373, 0.9326294, -2.0134086]
//Assign the current robot position shown in 6 angular
   numbers to V002 variable.
//Note: Elements in V variable are rounded to the nearest
    3 decimal places.
```

2.6.4 Get current joint position

```
GETPOS joint
```

Function: Get current joint position and assign it to a designated P variable.

Parameters: joint#1: joint position variable P, the range is 0~255. Joint coordinate variables that need to be converted

Example:

```
GETPOS P000
//To get the current joint coordinates J1, J2, J3, J4, J5
   , J6, and assign them to the P000 variable
```

2.6.5 Get a specific tool frame

```
GETTOOLFRAME pose TOOL#()
```

Function: Get a specific tool frame and save to a Cartesian Pose V variable Note: The user coordinate data obtained here is not the current user coordinate pose, but the corresponding check data of the user coordinate. It is necessary to open the V variable used to store data in the program before use.

Parameters: Pose: pose: spatial position variable V, size 0~255
TOOL#(): tool number, size: 0~7

Example:

```
GETTOOLFRAME V006 TOOL#(3)
//Read the tool#3 frame and save to V006
```

2.6.6 Set user coordinates

```
SETUSERFRAME USER#() pose
```

Function: Set user coordinates to V variable

Parameters: USER#(): the coordinate number that the user wants to set up, the range is 0~7
pose: spatial position variable V, range 0~255

Example:
```
SETUSERFRAME USER#(1) V001
```

2.6.7 Get a specific user frame

```
GETUSERFRAME pose USER#()
```

Function: Get a specific user frame and save to a Cartesian Pose V variable
Note: The user coordinate data obtained here is not the current user coordinate pose, but the corresponding check data of the user coordinate

Parameters: Pose: pose: spatial position variable V, size 0~255.
TOOL#(): tool number, size:0~7

Example:
```
GETUSERFRAME V009 USER#(5)
//Read the user#5 frame and save to V009
```

2.6.8 Set V variable to tool

```
SETTOOLFRAME TOOL#() V#()
```

Function: Set the value of the V variable to the tool, and set the value of the V variable to X, Y, Z, Rx, Ry, Rz of the tool coordinate system, indicating the position and posture of the tool.

Parameters: TOOL#(): select the tool number;
V#(): global variables

Example:
```
SETTOOLFRAME TOOL(0) V000
```

2.6.9 Set the current running tool number

```
SETTOOLNUMBER TF= toolNum
```

Function: Set the current running tool number

Parameters: toolnum: tool number, integer or B variable, integer range 0~7; B variable address range 0~255.

Example:
```
SETTOOLNUMBER TF=B0
```

2.6.10 Get the current running tool number to B variable

```
GETTOOLNUMBER B#()
```

Function: Get the current running tool number to the B variable

Parameters: B#(): The B variable used to store the tool number, the address range is 0~255.

Example:
```
GETTOOLNUMBER B0
```

2.6.11 Get tcp force

```
GETTCPFORCE V/LV [V/LV/ConstV/Tool#(N)/USER#(N)]
```

Function: Get the tool end force and store it in the first set of parameter V/LV variable.

Parameters: V/LV variable: end force data, range: V000-V255/LV000-LV255; [V/LV/ConstV/TOOL/USER]: reference coordinate system, optional. V/LV variable: system variable, range V000-V255/LV000-LV255; ConstV constant pose, double pose[6], x/y/z unit: mm, Rx/Ry/Rz unit: radian, range: [- π, π]; TOOL tool coordinate system, range: [0-7]; User user coordinate system, range: [0-7]. If this parameter is not entered, the default value is the end force of the current tool in base coordinates.

Example:
```
GETTCPFORCE V000 USER#(1)
GETTCPFORCE V010 TOOL#(2)
GETTCPFORCE V000 V001
GETTCPFORCE V000 ConstV=[1.1,2.2,3.3,-3.14,0,0]
GETTCPFORCE V000
```

Note: Currently, the end force can only be gotten under moving state, V/LV/ConstV/TOOL/USER are mutually exclusive.

2.6.12 Get joint torque

```
GETJOINTTORQUE P/LP
```

Function: Get joint torque and store in the P/LP variable.

Parameters: P/LP variable: joint torque, range: P000-P255/LP000-LP127.

Example:
```
GETJOINTTORQUE P000
```

Note: Currently, the joint torque can only be gotten under moving state.

2.6.13 Set the payload

```
SETPAYLOAD TOOL#(int) M=double X=double Y=double Z=double
```

Function: Set the mass and center of mass of the robot tool load. The decimal in the parameter keeps three decimal places

Parameters: TOOL#(): tool coordinate system number, range: 0~7 M: load kilograms, range:0~3.6 (EC63), 0~7.2 (EC66), 0-14.4 (EC612),
X,Y,Z: spatial position of the centroid offset, range:-5000~5000

Example:

```
NOP
SETPAYLOAD TOOL #(1) M=2.0KG X=10.0MM Y=20.0MM Z=30.0MM
//Set the load mass of robot tool number 1 to 2.0KG,
   center of mass X=10mm, Y=20mm, Z=30mm.
TIMER T=2.0
END
```

Note: This command is applicable to v2.13.2 and above.

2.6.14 Create user coordinate system

```
MFRAME USER# () joint#1 joint#2 joint#3
```

Function: Create user coordinate system command.
The user coordinates will be created by the given 3 points as definition points.

Parameters: UESR#(): custom user coordinates, range 0~7.
joint#1: joint position variable P, will be used as the origin of the user coordinate system.
joint#2: joint position variable P, will be used to determine the X+ direction of the user coordinate system.
joint#3: joint position variable P, will be used to determine the X-Y plan and Y+ direction of the user coordinate system.
Note: When the same three points P001, P002, and P003 in different orders and have different user coordinate systems calculated by MFRAME.

Example:

```
MFRAME USER# (1) P001 P002 P003
//Take P001 as the origin, P001 points to P002 as the
   positive direction of the X axis, and P003 is any
   point on the first quadrant of the XY interface except
    P001 and P002.
```

2.6.15 Get the actual TCP pose

```
GETACTUALTCP V/LV [TOOL#(N)] [USER#(N)]
```

Function: Get the actual tcp pose data in the base coordinate system or the specified user coordinate system

Parameters: V/LV: pose, standard parameters, range is V000-V255/LV000-LV255, double pose [6], the first three stand for position, unit of x, y, z is mm, range is [-1e+09~1e+09], the last three stand for pose, unit of Rx, Ry, Rz is radian, range is [-π, π]
TOOL#(N): tool coordinate system, optional parameter, N is the tool number, range is [0,7]. If it is not entered, the current tool is used
USER#(N): user coordinate system, optional parameter, N is the user number, range is [0,7]. If it is not entered, the base coordinate system is used, and if it is entered, the specified user coordinate system is used.

Example:
```
GETACTUALTCP V10 TOOL#(1) USER#(1)
```

2.6.16 Get the target interpolation TCP pose

```
GETTARGETTCP V/LV [TOOL#(N)] [USER#(N)]
```

Function: Get the target interpolation tcp pose data in the base coordinate system or the specified user coordinate system

Parameters: V/LV: pose, standard parameters, range is V000-V255/LV000-LV255, double pose [6], the first three stand for position, unit of x, y, z is mm, range is [-1e+09~1e+09], the last three stand for pose, unit of Rx, Ry, Rz is radian, range is [-π, π]
TOOL#(N): tool coordinate system, optional parameter, N is the tool number, range is [0,7]. If it is not entered, the current tool is used
USER#(N): user coordinate system, optional parameter, N is the user number, range is [0,7]. If it is not entered, the base coordinate system is used, and if it is entered, the specified user coordinate system is used.

Example:
```
GETTARGETTCP V11
```

2.6.17 Get the current actual joint

```
GETACTUALJOINT P/LP
```

Function: Get the current actual joint

Parameters: P/LP variable: joint, standard parameters, range is P000-P255/LP000-LP255, unit is radian, range is [-360,+360]

Example:
```
GETACTUALJOINT P10
```

2.6.18 Get the current target interpolation joint

```
GETTARGETJOINT P/LP
```

Function: Get the current target interpolation joint

Parameters: P/LP variable: joint, standard parameters, range is P000-P255/LP000-LP255, unit is radian, range is [-360,+360]

Example:
```
GETTARGETJOINT P11
```

2.6.19 Get the linear interpolation pose

```
GETINTERPPOSE V0/LV0 RATIO V1 V2
```

Function: Get the linear interpolation pose data between two given poses

Parameters: V0/LV0: output pose, standard parameters, range is V000-V255/LV000-LV255, double pose [6], the first three stand for position, unit of x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit of Rx, Ry, Rz is radian, range is [-π, π]
RATIO: output proportional value, standard parameter, floating-point data, range is [0,1]. When the value is equal to 0, the robot will return to the first pose. When the value is equal to 1, the robot will return to the second pose
V1: output pose, standard parameter, range is V000-V255
V2: output pose, standard parameter, range is V000-V255

Example:
```
GETINTERPPOSE V0 RATIO=0.5 V1 V2
```

2.7 Input and Output Instructions

2.7.1 The digital signal output instruction

```
DOUT OT# () switch|int|variable
```

Function: The digital signal output instruction. Assigns the specified state to a specific Y signal.

Parameters: OT#(): Digital output signal corresponding to Y variable.
May use one of OT#(), OG#(), OGH#(); OT#() takes the last digit, OGH#() takes the last four digits, OG#() takes the last eight digits. Size: Y0 to Y63.
switch|int|variable:rbitrary signal state, supported types are: integer, B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], ON=Enable, OFF=Invalid.
The current software version supports the assignment of decimal or floating-point numbers to the digital output signal, and the system does not report an error. The assignment logic is to round the input value, convert it to binary, and assign it to the digital output signal bit by bit; When assigning a negative number to a digital signal, the system does not report an error. The assignment logic is to take the absolute value of the negative number (decimals and floating-point numbers are also rounded), then perform the inversion and add one to the last bit, and finally assign the value to the number bit by bit. OT#() takes the last digit, OGH#() takes the last four digits, and OG#() takes the last 8 digits.

Example:

```
DOUT OT# (1) ON
// Output Y001=1
DOUT OT# (2) 9.5
// Round 9.5 and convert it to a binary number that is
   10=00001010, and take the last digit and assign it to
   OT#(2) that is Y002=0.
DOUT OGH# (1) 14
// Take the value of 14 and convert it into a binary
   number, which is 00001110, and assign the last four
   digits to OGH# (1), which is Y004=0, Y005=1, Y006=1,
   Y007=1
DOUT OGH#(1) -9
// The value of -9 is converted into a binary number,
   which is 00000111, and the last four digits are
   assigned to OGH#(1), which is Y004=1, Y005=1, Y006=1,
   Y007=0
DOUT OG# (1) -67.44
// Round -67.44 and convert it into a binary number,
   which is -67=10111101, and assign the last eight
   digits to OG#(1), which is Y008=1, Y009=0, Y010=1, Y012=1,
   Y013=1, Y014=1, Y015=1
```

2.7.2 Read the input signal

```
DIN variable IN# ()
```

Function: Read the specified input signal (X variable) and assigns it to the corresponding variable after converting it from a binary number to a decimal integer.

Parameters: variable: Global and local variables B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*] are supported; (see Section 2.1 for details).
IN#(): Digital input signal corresponding to X variable, accept one of IN#(), IG#(), IGH#(), size X0-X63. (see Section 2.1 for details).

Example:

```
DIN B001 IN# (1)
// Read X001 state and passed the value to B001
DIN I002 IGH#(2)
// Read X011~X008 state and converted the values to a
   decimal integer and pass it to I002. For example, X011
   ~X008=0101, then I002=5
DIN D003 IG#(3)
// Read X031~X024 state and converted the values to a
   decimal integer and pass it to D003. For example, X031
   ~X024=00101011, then D003=43
```

2.7.3 Analog signal output instruction

```
AOUT AO# () double
```

Function: Analog signal output instruction, assign the specified value to the specified AO signal.

Parameters: AO#(): Analog output signal, (see Section 2.1 for details).
double: Specify the output voltage value; the range is -10~10.
Note: When AO#(5), the double range is 0~10

Example:

```
AOUT AO# (1) 6.6
// Output 6.6 to AO#(1) signal
```

2.7.4 Get the analog input port voltage

```
AIN D001 AI# ()
```

Function: Get the voltage value of the analog input port and assign it to the corresponding variable

Parameters: Var_D : Global and local variables D, LD, D[*] or LD[*] are supported, (see Section 2.1 for details).
AI#():: Specific analog input interface to read, size 1 3; (see Section 2.1 for details).

Example:

```
AIN D004 AI# (1)
// Get analog input AI001 voltage and pass the value to
   D004
```

2.7.5 Virtual signal output instruction

```
MOUT M# () switch|int|variable
```

Function: The virtual signal output instruction assigns the specified state to a specific M signal.

Parameters: [M#()]: Virtual signal M, size M528 to M1471, accept one of M#() (size: 528 1471), MG#() (size: 66 183), MGH#() (size:132 367), MGD#() (size:33~91 U 300~447) (see Sec-tion 2.1 for details).
switch|int|variable: arbitrary signal state, supported type sare: integer, B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], ON=Enable, OFF=Invalid.
The current software version supports the assignment of decimal or floating-point numbers to the M signal, and the system does not report an error. The assignment logic is that the input value is rounded, converted to binary, and then assigned to the M signal bit by bit; When assigning a negative number to the M signal, the system does not report an error. The assignment logic is to take the absolute value of the neg-ative number (decimals and floating-point numbers are also rounded), then perform the inversion and add one to the last bit, and finally assign the value to the number bit by bit. Output signal; M#() takes the last digit, MGH#() takes the last 4 digits, and MG#() takes the last 8 digits.

Example:

```
MOUT M# (528) ON
// Output M#(528)=1.
MOUT M# (600) 11
// 11 is converted into a binary number, which is
   00001011, and the last bit is assigned to M# (600),
   which is M600=1
MOUT MGH# (135) -28
// -28 is converted into a binary number, that is,
   -28=11100100, and assign the last four digits to MGH
   #(135), that is, M540=0, M541=0, M542=1, M543=0
MOUT MG# (68) 135.59
// Round 135.59 and convert it to a binary number,
   136=10001000, and assign the last eight digits to MG
   #(68), which is M544-M546=0, M547=1, M548-M550=0, M551
   =1
```

2.7.6 Reads the specified M signal

```
MIN variable M# ()
```

Function: Reads the specified M signal and assigns it to the corresponding variable after con-verting it from a binary number to a decimal integer.

Parameters: variable: Global and local variables B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*] are supported; (see Section 2.1 for details).
M#(): Virtual signal M, accept one of M#() (range 0~1535), MG#( ) (range 0~191), or MGH#() (range 0~383); (see Section 2.1 for details).

Example:

```
MIN B001 M# (399)
// Read the input state of virtual input M399 and pass it
   to B001
MIN I002 MGH# (171)
// Read M687~M684 state and converted the values to a
   decimal integer and pass it to I002. For example, M687
   ~M684=0110, then I002=6
MIN D003 MG#(56)
// Read M455~M448 state and converted the values to a
   decimal integer and pass it to D003. For example, M455~
   M448=01011010, then D003=90
```

2.7.7 Pulse operation

```
PULSE OT# () T=double
```

Function: Impulse operation of specific signals

Parameters: OT#(): Digital output signal corresponding to Y variable. May use one of OT#(), OG#(), OGH#(). When the parameter is OG#(), all eight bits are set to 1, and when the parameter is OGH#(), all four bits are set to 1. Size: Y0 toY063.
T=double: pulse interval time (in seconds), accept float assignment

Example:

```
PULSE OT#(1) T=0.3
// Output Y001=1, after 0.3 output Y001=0
```

Note: The robot is in a waiting state when it executes the command.

2.8 Other Instructions

2.8.1 Save all P variables

```
SAVEVARP
```

Function: Save all P variables once
Note: The P variable will also be saved to the file in the cases of switching from automatic mode to other modes, emergency stop, end of automatic operation, and modifying the value of P variable on the monitoring interface.

Parameters: None

Example:

```
SETJOINT P001 -55.0671,-76.7208,87.6881,-84.3410,93.0787,
-0.4583
SETJOINT P002 20.2756,-71.7441,64.7525,-85.7218,
93.0779,-0.4606
SAVEVARP
```

2.8.2 Print command

```
TPWRITE string
```

Function: Print command. When robot execute this instruction, the specified content is recorded in the log file and displayed in the information window.

Parameters: string: Print content. Consist of numbers and letters

Example:

```
TPWRITE Completed
// Record "Completed" to log file and display on the
   information window
```

2.8.3 Clear instruction

```
CLEAR variable int
```

Function: clear any variable value and set it to zero

Parameters: variable: start address of an object variable, you can use one of B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*]; (see Section 2.1 for details).
int : the number of variables, can be either an integer or ALL. ALL means all variables of that type from the start address

Example:

```
CLEAR B000 3
//Clear variables B000.B001.B002 All values are set to
   zero
```

Note: It is impossible to clear the data of a certain axis of the variable P/V when executing the clear instruction. The data can only be cleared by using the SET command; the number of vari-ables should be greater than 0, and no error will be reported for negative numbers

2.8.4 Force control instruction

2.8.4.1 Start the force control mode

```
STARTFORCEMODE Mode=0 LV001|V001|ConstV LV002|V002|ConstV LV003|V003|ConstV
    LV004|V004|ConstV
```

Function: Command for starting the force control mode. The force control mode will be started when the robot executes this command.

Parameters: mode: mode, optional parameter, int[0,4]. The numbers mean the fixed mode, the point mode, the motion mode, the TCP mode and the pose mode. The default is 0.
arr_frame: the user-specified force control coordinate system, optional parameter, unit of x, y and z is mm, range is [-10000,10000], unit of Rx, Ry and Rz is radian, range is [-π,π]. The default is [0,0,0,0,0,0] and the force control coordinate system will be concentric with the TCP coordinate system.
arr_optional: force control method (mask of the DOFs), optional parameter, the value range is int[0,6] and the numbers mean the motion control, the force tracking, the fixed mode, the floating mode, the spring mode, the floating mode & motion control, the spring mode & motion control. The default is [0,0,0,0,0,0] and it indicates the motion control.
arr_torque: target torque, optional parameter, double arr_torque[6], the first three numbers mean the force, the torque range of EC63 is [-30,30]N, the torque range of EC66 is [-60,60]N, the torque range of EC612 is [-120,120]N and the torque range of EC616 is [-160,160]N; the last three numbers mean the torque and the range is [-1.5,1.5]N. The default is [0,0,0,0,0,0].
arr_speed: speed limit, optional parameter, unit is mm/s and °/s, double arr_speed[6], the first three numbers mean the linear speed, range is [0,200]; the last three numbers mean the angular speed, range is [0,11]. The default is [100, 100, 100, 5.73, 5.73, 5.73].

Example:

```
STARTFORCEMODE Mode=0 V001 V002 V003 V004
STARTFORCEMODE Mode=0 V001 ConstV=[1,2,3,4,5,6] V002 ConstV=
    [1,2,3,4,5,6]
STARTFORCEMODE Mode=0 ConstV=[1,2,3,4,5,6] V001 ConstV=[1,2,3,4,5,
    6] V002
```

Note: This command is applicable to v3.5.2 and above.

2.8.4.2 End the force control mode

```
ENDFORCEMODE
```

Function: Command for ending the force control mode. The force control mode will end when the robot executes this command. Please use the parameter together with STARTFORCEMODE.

Parameters: none

Example:
```
ENDFORCEMODE
```

Note: This command is applicable to v3.5.2 and above.

2.8.4.3 Clear all values of the force sensor to zero

```
ZEROFT
```

Function: Command for clearing all values of the force sensor to zero. It will record the current force sensor data when the robot executes this command.

Parameters: none

Example:
```
NOP
STARTFORCEMODE Mode=0 ConstV=[0,0,0,0,0,0] ConstV=[1,1,0,0,0,0]
ConstV=[0.000,0.000,0.000,0.000,0.000,0.000] ConstV=[100,100,100,
5.730,5.730,5.730]
ZEROFT
MOVEL V=200.0MM/S CR=0.0MM A=100MM/S² D=100MM/S² TOOL#(0) USER#(0)
ConstV=[354.3877720,186.5588535,551.5828166,-2.1964544,0.0957605,
-0.8161556] REF=[-17.088,-120.318,118.433,-91.828,93.481,34.990]
ENDFORCEMODE
END
```

Note: The current force sensor data will be read & saved when inserting the command after STARTFORCEMODE. In the process of the force control, the current readings will subtract what is previously saved. The command is only valid when selecting LUA, SDK and Back-end data source in the force control mode. The command is only supported in remote mode.
The command is applicable to v3.7.2 and above.

2.9 Lua Control Instruction

2.9.1 Run a specific script file

```
STARTLUA INDEX=int
```

Function: Run a specific LUA script file.
When using this instruction, you must first set up the script file in the script interface.

Parameters: INDEX=int: Index number of the specific script file. Range: 1~8

Example:
```
STARTLUA INDEX=1
// Run No.1 script file
```

2.9.2 Stop a script file

```
STOPLUA INDEX=int
```

Function: Stop a specific LUA script file

Parameters: INDEX=int: Index number of the specific script file. Range: 1~8

Example:
```
STOPLUA INDEX=1
// Stop No.1 script file
```

2.9.3 Reload a script file

```
RESTARTLUA INDEX=int
```

Function: Reload and run a specific LUA script file.

Parameters: INDEX=int: Index number of the specific script file. Range: 1~8

Example:
```
RESTARTLUA INDEX=3
// Reload and run No.3 script file
```

2.9.4 Get a script file state

```
GETLUASTATE B#(int) INDEX=int
```

Function: Get a specific script file state: 0=stopped,1=running.

Parameters: INDEX=int: Index number of the specific script file. Range: 1~8
B#(): the B variable designated to store the returned state code

Example:
```
GETLUASTATE B000 INDEX=1
// Get the state of No.1 script file and store the state
   code in B000
```

Chapter 3 JBI Script Examples

3.1 Subroutine call

```
//Subroutines: task01.jbi
NOP
MOVJ P001 VJ=10% PL=0
END
//Subroutines: task02.jbi
NOP
MOVJ P002 VJ=10% PL=0
END
//Main program: main.jbi
NOP
IF IN# (4) =1 THEN
//If the digital input signal X004= 1,
 CALL JOB: task01
//Calling subroutines task01
ELSEIF IN#(5) =1 THEN
// If the digital input signal X005= 1
 CALL JOB: task02
//Calling subroutines task02
ELSE
 DOUT OT#(6) ON
//Set digital output signal Y006=1
 WAIT IN#(6)=1
//Waiting for input signal X006= 1
ENDIF
END
```

3.2 Logical gate operation

```
NOP
SET B000 1
SET B001 2
SET B002 3
SET B003 4
TIMER T=3.0
```

```
AND B000 B001
//B0=0
AND B001 B002
//B1=2
AND B002 B003
//B2=0
TIMER T=3.0
SET B000 1
SET B001 2
SET B002 3
SET B003 4
TIMER T=3.0
OR B000 B001
//B0=3
OR B001 B002
//B1=3
OR B002 B003
//B2=7
TIMER T=3.0
SET B000 1
SET B001 2
SET B002 3
SET B003 4
TIMER T=3.0
NOT B000 B001
NOT B001 B002
NOT B002 B003
TIMER T=3.0
SET B000 1
SET B001 2
SET B002 3
SET B003 4
TIMER T=3.0
XOR B000 B001
//B0=3
XOR B001 B002
//B1=1
XOR B002 B007
TIMER T=3.0
END
```

3.3 Logical operation priority

t has been tested that multiple conditions connected by the logical operators &{and} and |{or} always follow the left-to-right principle in any scenario.

For example, A&B|C = A and B or C = (A B) C. Take the logic result of conditions A AND B, and then perform OR logical operation between the result and condition C to get the final.

A|B&C|D&E =(((A B) C) D) E: Take the OR logic result of conditions A and B, and then the AND logic result between it and condition C, and then the OR logic result between it and condition D, and then the AND logic result between it and condition E to obtain the final judgment result.

CAUTION

in the current software version, after all IF and UNTIL (including PAUSE IF, WAIT IF, MOVJ UNTIL and so on), if the variable types on both sides of the arithmetic symbols are not the same, the original value will not be processed according to the variable type (such as rounding, taking absolute or zeroing), and the system will not make any error report.

For example: IF I001=D003, so when I001=7, D003=7 .18, then this judgment does not hold.

IF B001=D002, in which case B001=0 and D004=-4, then this judgment is not valid.

IF OT# (5) = 17 in such that OT# (5) is ON, then this judgment holds. If OT# (5) is OFF at this time, this judgment is not valid.

3.4 Forward and inverse solutions

Positive solution: transform joint coordinates into rectangular coordinates; inverse solution: transform rectangular coordinates into joint coordinates

```
NOP
//Set the joint angle
SETJOINT P002 45,56,23,90,0,9,0.0000,0.0000
//Converts to Cartesian pose variable V001 via positive solution
   calculation
JOINTTOPOSE P002 V001
//Add 10mm to Z-direction
ADD V001(2) 10
// Converts the position to a joint position P002
POSETOJOINT V001 P002
//Run to point P002
MOVJ P002 VJ=100% PL=0
```

```
12 END
```

3.5 Calculation of coordinate system transformations

As shown in **Figure** 3-1 , The transformation matrix from coordinate system 2 to coordinate system 1 is V2, and the transformation matrix from coordinate system 3 to coordinate system 2 is V1. Then the transformation matrix from coordinate system 3 to coordinate system 1 can be calculated as V3.

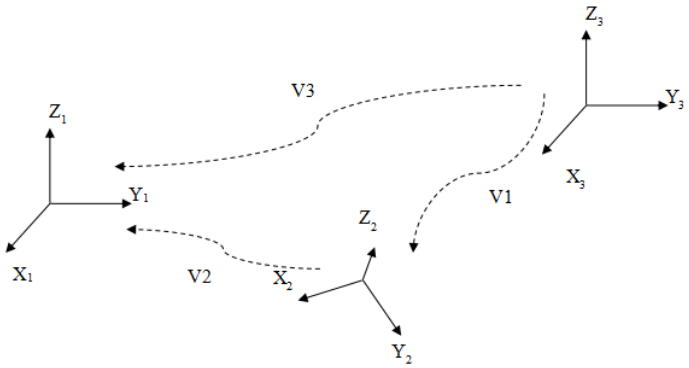


Figure 3-1 : Coordinate system transformation

```
1  NOP
2  //Set joint position
3  SETJOINT P002 45,56,23,90,0,9,0.0000,0.0000
4  //Converts to Cartesian pose variable V002 via forward solution
      calculation
5  JOINTTOPOSE P002 V002
6  // Assign V002 to V003
7  SET V003 V002
8  //Set up V001 variable
9  SETPOSE V001 0,0,200.00,0,0,1.5707
10 //Coordinate transformation, result store in V003 variable
11 POSEMUL V003 V001
12 // Converts the position to a joint position P002
13 POSETOJOINT V003 P002
14 //Move to P002
15 MOVJ P002 VJ=100% PL=0
16 END
```

ELITE ROBOTS
艾利特机器人

ALWAYS EASIER THAN BEFORE

Contact Us

Sales & Service: market@elibot.cn
Technical Support: tech@elibot.cn

Shanghai

Building 18, Lane 36,
Xuelin Road, Shanghai

Suzhou

1F, Building 4,
No 259 Changyang Street, Suzhou
+86-400-189-9358
+86-0512-83951898

Beijing

Room 1102, Building 6, No. 2,
Ronghua South Road, Beijing

Shenzhen

Room 202, Building 1A,
Hangkong Road, Shenzhen

WeChat
Official Account