CORE ROBOT

CL Programing Manual

2019. 6. 26

The information contained herein is the property of Core Robot, Inc., and shall not be reproduced in whole or in part without prior written approval of Core Robot, Inc. The information herein is subject to change without notice and should not be construed as a commitment by Core Robot, Inc.

This manual is periodically reviewed and revised.
Core Robot, Inc., assumes no responsibility for any errors or omissions in this document. Critical evaluation of this manual by the user is welcomed. Your comments assist us in preparation of future documentation. A form is provided at the back of the book for submitting your comments.

Copyright © 2016 by Core Robot, Inc. All rights reserved

CL Program Manual Revision History

Revision	Data	Comment	Remarks
1.01	16.10.11	Originally created	
1.02	18.03.21	Add instruction description	
1.03	18.04.02	Add SysTimer	
1.04	18.04.25	Modified SetP	
	18.05.09	Modified Speed	
2.00	19.06.26	coreCon version Release 4.0	
	19.08.16	Add #CvtJoint, Defined	

Content

1. Introduction 10
2. Data & Variables 11
   2.1. Numeric (Number) 11
      2.1.1. INTEGERS : 11
      2.1.2. REAL NUMBERS : 11
      2.1.3. LOGICAL VALUES : 12
   2.2. String 12
   2.3. Joint Position data 12
   2.4. Trans Position (Cartesian Coordinate) 13
   2.5. Array 15
   2.6. Local Variable and Subroutine Parameter 15
      2.6.1. Define Subroutine Parameter 15
      2.6.2. Set Local Variable 16
      2.6.3. Relationship between Subroutine and Local Variable 16
      2.6.4. Initialize variables 16
   2.7. Unit 17
3. Constant 18
4. Operator 19
   4.1. Unary Real Operators 19
   4.2. Binary Real Operators 20
5. Functions 21
6. Comment : ; 25
7. Controlling the program flow 26
   7.1. Goto 27
   7.2. CALL the other macro 27
   7.3. Signal Interrupt 28

7.4. Return, Stop, Pause 28
7.5. Wait 29
7.6. IF statement 30
7.7. Loop : While … Do, Do … Until, For 31
7.8. SWITCH … CASE DEFAULT END 32
8. Instructions 34
8.1. Transform & Position 34
8.2. Robot Motion 36
8.3. Robot Setting 39
8.4. I/O 42
8.5. Touch Probe 43
8.5.1. Example1 : Change Target point 1 45
8.5.2. Example2 : Target point scanning → Must be use the TouchWait. function 45
8.5.3. Example3 : Change Target point 2 46
8.5.4. Example4 : Individual acquisition Axis Data 46
8.5.5. Example5 : Read the Data through the SDO 47
8.6. Network Command 47
8.6.1. Client Mode 47
8.6.2. Server Mode 47
8.7. Conveyor Tracking 50
8.8. ETC 54
8.9. World Zone 54
8.10. ZComp 57
8.11. User Coordinate Instruction 58
9. Diagnostic Functions 60
10. Program selection with an external i/o 61
10.2. Example : Program selection with EPS MODE 63
10.3. Program selection with Bits instruction and SWITCH CASE structure 64

10.4. **calling a program using EPSWait conditional statement** 65
11. Example programs 66
11.1. Basic motion program 66
11.2. CP Motion 66
11.3. IO instruction 68
11.4. MoveC 69
11.5. Conveyor Tracking and Pickup 70
11.6. Base Move / Tool Move 72
12. Function Reference 73
12.1. Abs – Return absolute value. 73
12.2. ACos – Returns the arccosine value. 74
12.3. AIn – Returns the arccosine value. 75
12.4. AOut – Returns the output analog value. 76
12.5. Asc – get ASCII code 77
12.6. ASin – Return arcsine value. 78
12.7. Atan2 – return Arc tangent2. 79
12.8. Bits – Converts the value of bits to a number 80
12.9. CheckSum – Check the validity of communication data 81
12.10. $Chr – Converts Hex code to Ascii Char. 82
12.11. Cos – return Cosine value. 83
12.12. CvtTrans – Convert Joint position to Trans position. 84
12.13. Dest/#Dest – Return current motion's destination point 86
12.14. Distance – Return distance between A and B point. 87
12.15. DX, DY, DZ – X or Y or Z in Trans variable. 88
12.16. ErrorCode – Retrun User error code 89
12.17. Frame – Create coordinate frame. 90
12.18. $Fill– Assign a buffer to a string. 91
12.19. $HexStr – Convert hex code to string 92

12.20. Here/#Here – Assign the current location(Joint/Trans). 93
12.21. #Joint – Create a Joint position variable with Joint values for each axis. 94
12.22. JVal – Get a specific joint value of axis. 95
12.23. $Int16B – Converts Hex_code to 2-digit Ascii char 96
12.24. Len – Get length of string. 97
12.25. $Mid – Returns the specified string. 98
12.26. Random - Return a random value. 99
12.27. Round – Return rounded value. 100
12.28. Rx, Ry, Rz – Returns a rotating Trans matrix. 101
12.29. Sig – Return AND operation of the Sig1, Sig2, ... 104
12.30. Sin – Return Sin value. 105
12.31. Sqrt – Return Square root value. 106
12.32. SysTimer – Returns the SysTimer value corresponding to systimer id. 107
12.33. Timer – Returns the Timer value corresponding to timer id. 108
12.34. #TouchJoint/TouchTrans – Get TouchProbe detected postion 109
12.35. Trans – Create the Trans position. 110
12.36. Translate – Find the Trans matrix that moves point p by dx dy, dz. 111
12.37. Value – Convert String to number 112
13. Instruction Reference 114
13.1. ABOVE/BELOW – Robot ABOVE and BELOW configuration. 114
13.2. Accel/Decel – Set acceleration and deceleration. 115
13.3. Accuracy – Sets the position reach recognition range. 116
13.4. ALIGN – Align the z direction of the tool with nearest Base coordinate axes. 117
13.5. ApproJ – Joint move a distance with -Z direction of the tool coordinate system 118
13.6. ApproL – Linear move a distance with -Z direction of the tool coordinate system 119

13.7. Brake – Stop Robot motion and proceed to the next step.....120
13.8. Break - Break the CP motion and move it to the correct position.....121
13.9. Call – Call program.....122
13.10. Delay – Delay previous command for setting time.....123
13.11. DelayTime – Wait until the next command.....124
13.12. DelayDO – Delay Digital Output for setting value.....125
13.13. DepartJ – Robot moves -Z direction in the tool coordinate system.....126
13.14. DepartL – Robot moves linearly -Z direction in the tool coordinate system 127
13.15. Drive – Moves the selected axis.....128
13.16. FMoveL – Fixed FTool works with uniform distance.....129
13.17. FTool – Set the FTool(Fixed Tool).....130
13.18. HALT – Stop the Program.....131
13.19. HOME/HOME2 – Moves specified Home position.....132
13.20. IncJ – Moves by set Joint.....133
13.21. IncL – Moves linear interpolation for the base coordinate system.....134
13.22. IncT – Moves based on Tool coordinate.....135
13.23. LEFTY/RIGHTY – Select the robot geometry.....136
13.24. MakeStr – Make String variable.....137
13.25. MoveC – Moves the robot circularly.....138
13.26. MoveH – Pass the Singular position in linear movement (Only Serial6 robot). 139
13.27. MoveJ – Moves the robot by Joint movement.....140
13.28. MoveL – Moves the robot linearly.....141
13.29. MoveX – Monitor the specified signal during the MoveI motion.....142
13.30. MoveXJ – Monitor the specified signal during the Movej motion.....143
13.31. MovePi, MoveP – Move multiple points in constant velocity.....144
13.32. MSig – IO signal is output during Motion step.....145
13.33. OverDI – Control the digital input signal.....147

13.34. PrefetchSig – Contral the PREFETCH_SIGNAL command. 148
13.35. Reset – Turn off all the output signal. 149
13.36. RunMask – Option to export signal only at running process. 150
13.37. SetAICalib – Set the calibration table about Analog input channel 151
13.38. SetAO – Output the Analog signal. 153
13.39. SetAOCalib – Set the calibration table about Analog output channel .. 154
13.40. SetDO – Digital Output. 156
13.41. SetJ – Set the Joint variable data. 157
13.42. SetP – Set the variable Trans value. 158
13.43. SClose – Close the serial communication. 159
13.44. SOpen – Open the serial communication. 160
13.45. SRead – Read Serial data. 161
13.46. SRead2 – Read serial data of selected section. 162
13.47. SWrite – Write serial data. 163
13.48. SINGLE/DOUBLE – Determine the rotation range of the 6 axis. 164
13.49. Smooth – S-curve motion. 165
13.50. Stable – The robot stick the specified position for the set time. 166
13.51. StableTime – Use the Stable function for motion commands. 167
13.52. SPEED – Set the operation speed. 168
13.53. SubAbort – Stop the Subtask. 170
13.54. SubExecute – Execute Subtask. 171
13.55. UWRIST/DWRIST – Determines the shape of WRIST in ROBOT 172
13.56. Wait – Waits until the condition is satisfied or the set time is over. 173
13.57. WaitTime – Waits for the entered time. 175
13.58. WaitSig – Wait until the condition is satisfied 176

1. Introduction

CL(Core Language) Robot Language provides the high level of functionalities to write a robot program for the various applications. This is a reference manual containing a detailed explanation of the programming language as well as all data types, instructions and functions.
If you want to learn about how to operate with the coreCon, “coreCon User’s Guide” will help you.

2. Data & Variables

CL has four data types which are numeric, joint location, trans location, and string. In addition to those basic types the array of each data type can be used. The data are used as the arguments of functions and instructions, and are defined as the variables or the constants.
The variable name is defined with the combination of alphabetic characters and the decimal numbers. However, it cannot be started with the number and have the special character, either. For example, “V100” is a right variable name, but “100V” or “V@100” is wrong name. The dot (.) and under score (_) as well as the alphabetic characters can be used as a variable name. Therefore V.temp and V_press are valid names. The system reserved keyword cannot be used as a variable name such as if, random. To distinguish the actual type of variable, string variable and joint location variable starts with special character. The string variable starts with $ and the joint location variable starts with #. The functions related with those data types also start with the same special characters.

- Macro program name length is 19 characters long and variable is 20 characters
- Joint, Trans variable maximun size are 100Byte, Number variable maximun size is 24Byte, String variable maximun size is 220Byte
  Total variable storage box size is 8Mbyte.
  Therefore, when creating variable 8 Mbyte, the remaining variable may not be created.

2.1. Numeric (Number)

Numeric data is a combination of numerals, variables, operators, and functions which return numeric values. Numeric expressions are used not only for mathematical calculations, but also as arguments for monitor commands or program instructions. Numeric values used in the CL system are divided into the three types described below.

2.1.1. INTEGERS :

Integers are values without fractional parts (whole numbers). Values with full precision ranges are from -16,777,216 to +16,777,216. Values that exceed this range are rounded to seven significant digits. Integer values are usually entered as decimal numbers, however, it may be more convenient to enter them in binary or hexadecimal notations. The hex number is set by adding ^H in front of the digit.

2.1.2. REAL NUMBERS :

Real numbers have both the integer part and a fractional part which can range from -3.4E +38 ~ 3.4E +38. Like integers, real values are positive, zero or negative. They can be represented in scientific notation. Real values are stored with an accuracy of approximately seven digits, but actual values may have less precision caused by a calculation error.

2.1.3. LOGICAL VALUES :

Logical values have only two states, ON or OFF. These two states are also referred to as TRUE and FALSE respectively. A value of negative one (-1) is assigned for the TRUE or ON state and a value of zero (0) is assigned for the FALSE or OFF state.

```
volume = 100
vol2 = -10.3
vol3 = 1.2e+4
flag = ^H6BA3
```

Currently, CL does not distinguish between integer type and real type, only real type is used.An integer is a number from which a decimal point is removed from a real number.

2.2. String

String data is enclosed by the double quotation mark. It represents the character array. String variable starts with the $ to distinguish form others.

```
$name = "Robert Kim"
$msg = "Program completed"
```

String add operation

```
$test1 = "K"
$test2 = "im"
$test3 = $test1 + $test2 = "Kim" (Only same variable type)
```

2.3. Joint Position data

A joint position value is represented by the exact position of the individual robot joints in degrees. There are several characteristics of joint locations that should be considered. These characteristics result from joint angles being recorded.

- Advantages of joint position : Repeatalbility precision is achieved and there is no ambiguity about robot configuration at a location.

- Disadvantages of joint locations : The values recorded can be used by any model of robot, however the tool center point location is different when used by a robot of different physical size. Precision locations cannot be easily modified to compensate for location changes in the robot workspace, because a change requires complete knowledge of the relationship between the positions of all robot joints and the locations in the robot workspace.

Joint position data must start with '#' variable name to distinguish it from orthogonal position data. Position data can be stored in Teaching through the current position and can be defined as any value you want using a function called Joint().

```
MoveJ #p1 ;   Joint Move to joint location #p1
MoveL #p2 ;   Joint move to joint location #p2

Point #p3 = Joint(10, 20) ; Define the joint location #p3
Point #p4 = #p3 ; Copy the joint location #p3 to #p4

delta = 5
Point #p5 = Joint(delta, delta*2, -delta*2) ; Joint location using the numeric variable
```

Refer the above example, the location variable is defined by Point location_variable = target_value. The actual value of #p3 location is (10,20, 0, 0, 0, 0). The default value is set to 0 for the missing arguments.

2.4. Trans Position (Cartesian Coordinate)

A transformation position is represented by defining the location in terms of a Cartesian (XYZ) reference frame fixed to the base of the robot. The position of the tool center point is defined with X, Y, and Z coordinates, and the tool orientation is defined by three Euler angles measured from the coordinate axes. The Euler angles are represented as (A, B, C). To differentiate from the joint location variable, the trans location variable has no prefix character

- Advantages of transformation position : A value defined for use with one robot can be used with a different robot having a similar work envelope because the value is defined in terms of workspace coordinates. Transformations are easily modified to change a location within the robot workspace. A powerful feature of transformation locations is the ability to define locations as combinations of values. This is called compound or relative transformation. Such values are used to define the location of a part relative to its fixturing.

- Disadvantages of transformation position : Since a transformation location defines the location of the tool center point in terms of coordinates in the workspace, no information is provided about the specific robot configuration at the location. Whenever a transformation is used to define the destination of a robot motion, the AS system must convert the transformation location into an equivalent precision location so it knows how to move the individual joints. This conversion can introduce small location errors. Despite these disadvantages, transformation locations are generally much more convenient than precision locations.

```
MoveJ pt1 ; Move to trans location pt1
MoveL pt2 ; Move to trans location

Point pt3 = Trans(10, 20) ;   Define trans location variable pt3
Point pt4 = pt3 ; Copy the translocation variable pt3 to pt4

delta = 5
Point pt5 = Trans(delta, delta*2, -delta*2) ; Define trans variable using numeric variable
```

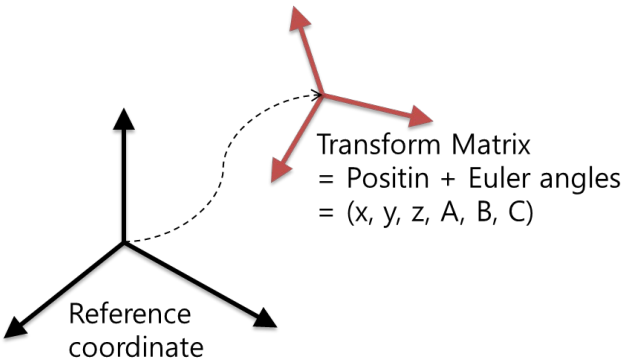


Figure 1 Trans location data

The orthogonal position data is defined by the reference coordinate system.The actual position changes according to the reference coordinate system.Therefore, it should be noted that the position of the robot can be changed when changing the reference coordinate system.

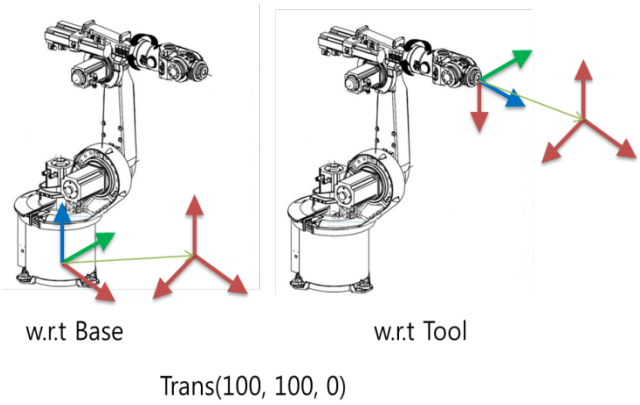


Figure 2 Actual location depends on the reference coordinate

Even the same location Trans(100, 100, 0), the actual location will be different by the reference coordinate. But as the joint location defines the each axis position, it forms the same pose for all cases if the robot is the same robot

2.5. Array

An array is a group of values that share a single name. Location variables can be scalars or arrays. A location scalar is a single location value. Each value in an array is called an element of the array. An element of a location array is specified in exactly the same way as an element of a numeric array by appending an index enclosed in brackets to the array name. For example, "part[7]" refers to element 7 of the array "part." Indexes must be integers in the range of 0 ~ 9999. Three examples of arrays are described.

```
A[1] = 10 ; numeric array
A[2] = 20
$name[0] = "John" ; String array
Point #p1[0] = Joint(10,0,3) ; location array
```

2.6. Local Variable and Subroutine Parameter

2.6.1. Define Subroutine Parameter

At the beginning of the program, define the variable to be used as the .PARAM Argument as shown below.

Program 1: main(Macro Program name: main)

```
.PARAM $retstr
.LOCAL $strval
$retstr = ""
$strval = "Core"
Call prog($strval, $retstr) ;call prog() program, input: $strval, output: $retstr
TPWrite 2,"retstr=%s",$retstr
```

program 2: prog (Macro Program name: prog)

```
.PARAM$str, $retstr
.LOCAL $local_str
$local_str = "Robot"
$retstr = $str + $local_str
```

2.6.2. Set Local Variable

If you want to define a variable that is valid only within a Subroutine, you can use the .LOCAL Argument command to define the variable.

```
.LOCAL local_num1, $loval_str1, #local_p1, local_p1
```

When calling a variable, the program variable is not used when there is a variable with the same name as the program variable, because it always looks for the LOCAL variable first.

2.6.3. Relationship between Subroutine and Local Variable

Since the Subroutine Parameter is also used as the LOCAL variable, it has the same properties as the LOCAL variable. However, since the variable entered as a parameter is passed to Reference, the data changed in Subroutine is changed to the input variable data. In the above example, if you change the contents of $ strvar1, which corresponds to prog, $str in 2nd program(prog) will be changed.

2.6.4. Initialize variables

```
.LOCAL #joint1, trans1, num1, $str1, numarr[0]
Point #joint = joint(0,0,10)
Point trans1 = Trans(0,0,10)
num1 =10
$str1 = "string"
numarr[0] = 1
numarr[1] = 2
```

2.7. Unit

The default units are as follows.

- Length/Distance : mm
- Angle : deg
  - Angle unit is degree. The argument value used in Sin(30) is not radian but degree.
- Velocity and Acceleration : %
  - The percentage with respect to the maximum velocity and acceleration.
  - The maximum velocity is defined in the robot configuration. In case the maximum velocity is 3000mm/s and the Speed is 10, the actual speed will be 300mm/s (= 3000mm/s * 10%).

3. Constant

CL has the predefined constant variable It helps the exact representation in some instructions.

- Unit constant
  - Length(Distance) : mm
  - Time : s
  - Speed : mm/s, sec, mm/min
- on / off : ON, OFF
- true/false : TRUE, FALSE
  - Internally the value of ON/TRUE is -1, OFF/FALSE is 0.
- PI : 3.141592…
- NULL : identity trans matrix
- Icon Contant
  - ICONE : Error Icon = 0
  - ICONW : Warning Icon = 1
  - ICONI : Information Icon = 2
  - These constants is used TPWrite instructions. The given icon is displayed in Message icon.

```
Speed 80 mm/s
Speed 5 sec ; Move to the target for 5 seconds
WaitTime 1.5 s ; Wait 1.5 seconds
A = ON ; A is -1
B = OFF ; B is 0
```

4. Operator

4.1. Unary Real Operators

- COM : Complement
- - : Negative
- NOT

```
A = -3
Aa = -A ; Aa becomes 3

B = COM Aa ; Aa becomes 0xFFFFFFFC

C = TRUE
D = NOT C ; D becomes FALSE.
```

4.2. Binary Real Operators

- ^ : square,
- * , / , MOD : multiplication, division and remainder
- +, - : addition and subtraction
- <, <=, =<, ==, >=, =>, > : Relational operators
- BAND, BOR, BXOR : Bit operators
- AND, OR, XOR : Logical operators

```
a = x^4 + 3 * (3 – x) ;
if a AND b then … ;
if a <= b then … ;
if (a < b) AND (c > d) then … ;
Wait Sig(1001) OR Sig(1002) ;
```

5. Functions

A function is a command with a return value, which is used in the form return_value = function (args ...). If there is no argument, omit () and write only the function. The argument [] in the argument list is the default value, if it is omitted as an optional argument.

The table below is a list of functions to be provided. Please refer to the description section for details.

Function	Arguments	Details
Abs	Number x	Returns absolute value of the argument. A = Abs(-3.5) → A = 3.5
AOut	Number channel	Returns the output analog value. A = AOut(2) → Analog output value of channel 2
AIn	Number channel	Returns the input analog value. A = AIn(2) → Analog input value of channel 2
Atan2	Number Y Number X	Returns the arctangent value(y/x) The result is in degrees. A = Atan(1, 1) → A = 45
Asc	String s [Number index = 1]	Returns the ascii value of the character corresponding to the index of string s. A = Asc("sport", 2) → Assign the ascii value of the second letter 'p' in "sport". Return first character value, if no argument.
Bits	Number start_signal Number count	Reads a consecutive number of signal values from the start signal number. . Ex)1007 = on, 1006 = on, 1005 = off, 1004 =on A = Bits(1004, 4) → 1011(B) ,return 13 .
CheckSum	Strings, Type	Check the validity of communication data, in serial communication. chkvalue = CheckSum($data, type) Type: 0: CRC16 1: VRC(Vertical Redundancy Check, XOR) 2: LRC(Longitudinal Redundancy Check, XOR) 3: SUM(sum) 4: OR(or)
$Chr	Hex hex_code	Converts Hex code to Ascii Char.

		hex_code = ^h31 $char = $Chr(hex_code) ; '1'
Cos	Number X	Return cos value of x. argument x unit is degree A = Cos(90) → A = 0
#CvtJoint	TransPoint A [JointPoint B] [IsLefty C] [IsBelow D] [IsUWrist E]	Convert Trans position to Joint position. B,C,D,E are Option. C,D,E are robot configuration option. Point #p1 = #CvtJoint(pt1) : default init Joint = (0,0...) Convert the Trans position pt1 to Joint position #p1 based on Joint(0,0,...) Point #p2 = #CvtJoint(pt2, #pinit, FALSE, FALSE, FALSE) Convert the Trans position pt1 to Joint position #p1 based on #pinit. Because Robot configuration optioins are all FALSE, so robot configurations are set to isRighty, isABOVE, isDWrist.
CvtTrans	Point #joint	Convert Joint position to Trans position. Point ptrans = CvtTrans(#pjoint)
Dest/#Dest		Return current motion's destination point. Point pold = Dest / Point #pold2 = #Dest
Distance	Position A Position B	Return distance between A and B point. A = Distance(p1, p2)
DX, DY, DZ	Trans p	Return x,y,z value of point p. Xval = DX(pt1), Yval = DY(pt1), Zval = DZ(pt1);
ErrorCode	ErrorCode	Return errorcode a = ErrorCode
Frame	Position porg Position px Position pxy Position pz	Create coordinate frame. The origin of the coordinate system is porg, the x vector becomes px. Define the plane in which the y-axis with the point Pxy, the z-axis direction is determined by pz. Point newcoord = Frame(porg, px, pxy, pz)
$HexStr	String s	Convert Hex code to string. $char = "12" $hex_str = $HexStr($char) ; "3132"
Here/#Here		Assign the current location. Point pcur = Here Point #pcur_j = #Here

#Joint	Number j1 Number j2 …	Creates a joint position variable with joint values for each axis . Point #pjnt = Joint(0, -20, 10) All omitted arguments are assigned 0. Point #porg = Joint(0) → Assign all points with zero.
$Int16B	Hex hex_code	Converts Hex_code to 2-digit Ascii char hex_code = ^h3161 $hex_str = $Int16B(hex_code) ; "1a"
Len	String s	Returns string s length. A = Len("sport") → A = 5
$Mid	String s Number index Number count	Returns the number of strings from the position corresponding to the index of the string s. $A = $Mid("sport", 2, 3) → A = "por"
Random		Return a random value Range(0 ~ 1). A = Random → A = random value of 0~1.
Round	Number x	Return rounded x A = Round(3.5) → A = 4
Rx, Ry, Rz	Number x	Returns a rotating Trans matrix, each argument rotated by x degrees. Point trx = Rx(30) → Trans matrix that rotates x axis by 30 degree
Sig	Number sig1 Number sig2 Number …	Return and operation of the Sig1, Sig2, … Ex) 1002 = on, 1003 = off A = Sig(1002, 1003) → A = 0(FALSE) 1002 = on, 1003 = on A = Sig(1002, 1003) → A = -1(TRUE)
Sin	Number x	Return Sin value, argument x unit is degree A = Sin(90) → A = 1
Sqrt	Number x	Return sqrt vlue, argument x. A = Sqrt(4) → A = 2
SysTimer	Number x	Returns the SysTimer value corresponding to timer id. usec = SysTimer(1,1) → return usec of systimer 1 sec = SysTimer(2,2) → return sec of systimer 2 day = SysTimer(3,3) → return day of systimer 3
Timer	Number id	Return Square root value A = Sqrt(4) → A = 2

#TouchJoint TouchTrans	Number index	Returns the Timer value corresponding to timer id. A = Timer(1) → return timer 1 value
Translate	Position p	The position value is received when the state of the touched signal corresponds to the index. A has the current position when the external signal is detect. Find the Trans matrix that moves point p by dx dy, dz . If no argument, set to 0. A contains the value obtained by moving the point P by 10mm for the x-axis, 2mm for the y-axis, and 0mm for the z-axis.
	Number dx	
	Number dy	
	Number dz	
Trans	Number x	This function creates the Trans position variable by directly setting the value of the trans point. Point p = Trans(0, -20, 10, 20, 20, 10)
	Number y	
	Number z	
	Number A	
	Number B	
	Number C	
Value	String s	Convert String to number A = Value("12") → A = 12

6. Comment : ;

Comment starts with " ;" The behind step string is ignored on running.

```
; Move to wait position
MoveJ #pwait
WaitSig  1001
; Start to work
MoveL #ps
```

7. Controlling the program flow

This chapter introduces the structures available in CL. CL provides the most control structure instructions including a branch and looping. In addition CL also provides the specialized instructions in robot application.

- Goto
- If condition goto label
- If condition then ··· Else ··· End
- While condition Do ··· End
- Do ··· Until condition
- For to step ··· End
- Switch . Case Default
- Call program
- Interrupt Signal#, InterruptHandler, [Motion Stop = TRUE]
- Return
- Pause
- Stop
- Wait condition, [timeout], [result]
- WaitTime time

7.1. Goto

The GOTO instruction causes program execution to branch immediately to a program label instruction somewhere else in the program.

- GOTO label [IF condition] :
- IF condition Goto label.

label is an integer entered at the beginning of a line of program code. label is not the same as the program step numbers: Step numbers are assigned by the system; labels are entered by the programmer as the opening to a line of code.

```
100     MoveJ #ptmp
        IF Sig(1001) Then
                Goto 100            ; goto Label 100
        END
```

For simple condition, Goto can be used together with if clause.

```
IF Sig(1001) Goto 100
Goto 100 IF n > 3
```

7.2. CALL the other macro

- CALL subroutine

  CALL instructions are used to implement subroutine calls. The CALL instruction causes program execution to be suspended and execution of a new program to begin. When the new program has completed execution, execution of the original program will resume at the instruction after the CALL. The subroutine name used in CALL is a program name. It is required to make another program to use as an argument of CALL instruction.
- Call subroutine(arg1,arg2)

  Argument can be used when calling another program with Call command.
- Calling the same subroutine at the same time in each task is prohibited.

```
; Prog1
        MoveJ #pdrop
; Prog2
        MoveJ #ppick
        SetDO hand2
; Prog3
            If Sig(1001) then
                    Call Prog1
            Else
                    Call Prog2
            End
```

7.3. Signal Interrupt

- **Interrupt** Signal#, InterruptHandler
- **Ignore Signal# ; Terminate the corresponding signal interrupt**

  CL provides a interrupt handler. A program can be interrupted based on a state transition of a digital input signal. When the watching signal is changed, the robot motion can be stopped immediately or finish by option parameter.

```
; Prog1
        MoveJ #pdrop
        SetDO hand1
        Interrupt 1002, Prog2 ;  If the digital input signal 1002 is changed to On,the
 Prog2 will be executed after stopping.
; Prog2   Interrupt Handler
        MoveJ #ppick
        SetDO hand2
```

7.4. Return, Stop, Pause

- **RETURN :** The execution of current program is ended and return to the caller subroutine if the current program is called by it.
- **STOP :** The execution of the current program cycle is terminated and the next execution cycle resumes at the first step of the program.
- **Pause :** The execution of the current program is paused and the robot changed to hold state. The execution can be resumed by run operation.

In this example, ProgMain is a main program executed at first. In this program the subroutine Prog1 and Prog2 is selectively called by the input signal 1001. If the Prog2 is called, the program goes to the first step of ProgMain due to the Stop instruction.

```
; ProgMain
        If Sig(1001) then
            Call Prog1
        Else
            Call Prog2
         End
; Prog1
        MoveJ #pdrop
      SetDO hand1
      IF condition1   == 1 Then
            Call Prog2
        Else
        IF condition2 Then
              RETURN ; returns to the caller program
        END
      END
      Pause ; Robot motion is stopped and the program can be resumed by user operation.
; Prog2
      MoveJ #ppick
      SetDO hand2
      Stop ; All remained program instructions are canceled and goes to the first step of MainProg.
```

7.5. Wait

- WAIT condition, [Timeout], [Result]

WAIT suspends program execution until a condition (or conditions) becomes true. If the optional argument, timeout is set, the wait is ended even though the condition does not meet. You can distinguish the result of actual condition checking the result argument. If the result is TRUE, it means that the wait condition is TRUE.

```
    Wait Sig(1001)
; Wait forever until the digital input signal 1001   is on
    Wait Sig(-1001, 1002), 2, result
; Wait for 2 seconds if two digital input signal does not meet. Or, ends the instructions
immediately if signal 1001 is off and 1002 is on. The result of condition is assigned
to result variable.
    If result then ;
        TPWrite ICONI, "Job succedded"
    Else
        TPWrite ICONI, "Timeout"
```

Reference. SIG(1001, -1003) : The AND operation is applied to the conditions of two signals. If the OR condition is needed, the statement can be written like this.
Wait Sig(1001) OR Sig(-1003)
Other various conditions as well as the signal condition can be used

```
    Wait Timer(1) > 100
    Wait n > 100
```

- **WaitTime** duration_second

Suspend the program execution during the given time. The unit of time is second.

```
val = 2.5
WaitTime 0.5
WaitTime val
```

7.6. IF statement

The basic conditional instruction is the IF...THEN...ELSE clause. This instruction has two forms. Each instruction must be placed one line at a time.
(You can use up to 10 times.)

```
If   n > 5 THEN
          sp = 50
ELSE
          sp = 70
END
```

중첩된 If, Else If 문은 아래와 같이 Else 안에 다음 IF 문을 정의해야 합니다.

```
    IF m THEN
        IF n THEN
        ELSE
        END
    ELSE
        IF n2 THEN
        ELSE
            IF n2 THEN
            END
        END
    END
```

7.7. Loop : While ··· Do, Do ··· Until, For

CL provides most commonly used looing structure instructions. These instructions allow you to execute blocks of code a variable number of times.

- While condition Do : WHILE...DO is a looping structure similar to DO...UNTIL except the boolean expression is evaluated at the beginning of the loop instead of at the end. This means that if the condition indicated by the expression is true when the WHILE...DO instruction is encountered, the code within the loop will not be executed at all.
- Do ··· Until condition : DO...UNTIL is a looping structure that will execute a given block of code an indeterminate number of times. Termination of the loop depends on the Boolean expression or variable that controls the loop becoming true. The boolean is tested after each execution of the code block—if the expression evaluates to true, the loop is not executed again. Since the expression is not evaluated until after the code block has been executed, the code block will always execute at least once.
- **FOR** loop_variable = initial **TO** last [**STEP** increment] : Default STEP increment is 1
- A FOR instruction creates an execution loop that will execute a given block of code a specified number of times

```
While TRUE Do
        MoveJ #p1
        Goto 200 IF condition
END
200 TPWrite 2, "While ended"
```

```
Max.row = 5
Max.col = 5
FOR row = 1 TO max.row
        POINT hole = Translate(start.position, (row-1)*100, 0, 0)
        FOR col = 1 TO max.col
                CALL pick.place ; update next position
                POINT hole = Translate(hole,   0, 100, 0)
        END
END
```

7.8. SWITCH … CASE DEFAULT END

The SWITCH structure will allow a program to take one of many different actions based on the value of a variable. The variable used must be a real or an integer. The form of the SWITCH structure is

```
SWITCH   numeric_value
CASE   case_value11, case_value12, … :
 Instructions
CASE   case_value21, case_value22, …:
       Instructions
DEFAULT:

END
```

```
Point #p1 = Joint(0)
Point #p22 = Joint(10)
Point #p3 = Joint(20)
Point #p4 = Joint(30)
Point #p5 = Joint(40)
MoveJ   #p1
FOR i = 0 TO 4
        TPWrite 2,"Case : %d ",i

        SWITCH i
        CASE 0,1 :
                MoveJ   #p22
        CASE 2 :
                MoveJ   #p4
        default:
                MoveJ   #p5
        END
END
```

8. Instructions

Instructions have a different syntax from functions. They have no return value and no parenthesis to list the arguments. Some instructions must pass the variable argument to get a result from inside execution. CL provides the robot motion command, IO settings and waiting, and communications command as the instructions. For the details of each instruction, refer to the reference parts of the manual.

8.1. Transform & Position

Before going on the motion control instructions, the concepts of transformation and its operation are figured out. CL's trans location variable is a homogeneous transformation as it mentioned before. The most important operation on the homogeneous transform is matrix multiplication as it means added location shift or rotation. CL provides the forward transformation and inverse transformation as a operator + and -.

Trans variable operations: + / -

Trans location variable is the homogeneous transform matrix. Internally it is 4 by 4 matrix, but displays as (X, Y, Z, A, B, C). (A, B, C) are Euler angle representing a rotational part. Therefore, the operator + means the matrix multiplication and – operator means the matrix multiplication with inversed matrix. For example, trans location variables Pa, Pb are given, two operations are as follows.
Pc = Pa + Pb → Pc = Matrix Pa * Matrix Pb
Pc = Pa – Pb → Pc = Matrix Pa * Matrix Pb^-1

Trans When the position variables Pa and Pb are present, Pc = Pa + Pb operation becomes Trans Multiplication.Also, Pc = Pa - Pb operation is the result of multiplying Pa by Pb inverse transform matrix.

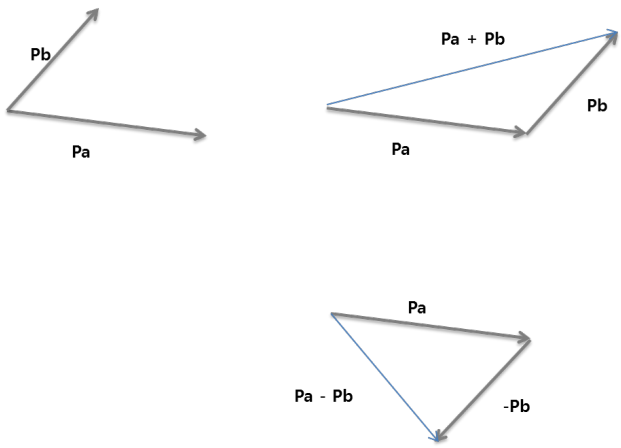


Figure 3 The operations of location variables

```
Point #location2 = #Here ; Save the current location to a location variable
POINT    location1 = location2    ; Assignment
POINT #place = #post

POINT pick = corner + pick ; Add operation
```

To modify the location variable use Point loc_var1 = loc_var2 for trans location variable or Point #pj1 = #pj2 for joint location variable. For the assigment operations, the joint location variable cannot be assigned to the trans variable. To get the trans location variable from the joint variable, use the CvtTrans function

- DECOMPOSE x[0] = part
- DECOMPOSE angle[4] = #pick

DECOMPOSE gets the each component of location variable as an array. For the trans location variable, X, Y, Z, A, B, C value of trans location variable copied to array element. For the joint location variable, each joint value is copied..

```
Point #p1 = Joint(100, 0, 10, 5)
DECOMOSE x[0] = #p1
x[0] = 100
x[1] = 0
x[2] = 10
x[3] = 5
```

- TOOL tfname
- BASE tfname

The current tool coordinate and base coordinate is changed with TOOL and Base instructions. Tool and Base frame is the same type as the trans location variable. You can set the tool and base frame with various methods in coreCon menu function.

8.2. Robot Motion

- MoveJ Position → Abbreviation: MJ : Joint interpolation motion
- MoveL Position → Abbreviation: ML : Straight line interpolation motion.
- MoveC Position 1, Position 2 → 약어 : Abbreviation: Circular interpolation motion.
- MoveX Position, Signal → Abbreviation: MX

M

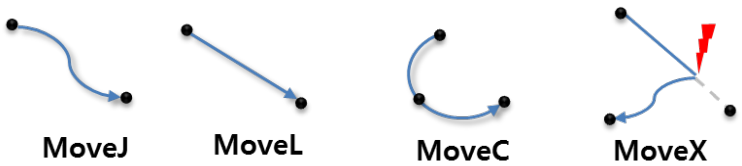


Figure 4 Robot Motion Instructions.

With CL, a motion instruction such as "MoveJ #p1" is interpreted to mean start moving the robot to location '#p1'. As soon as the robot starts moving to the specified destination, the CL program continues without waiting for the robot motion to complete. The instruction sequence.

```
MoveJ #p2 ; Robot move to position #p2 in joint motion
SetDO 2 ; Output the digital signal 2
MoveJ #p3 ; After finish MoveJ #p2 command, robot move th position #p3 in linear motion
```

- Delay time
- Stable time

Delay and Stable are motion instruction. Delay and stable instructions after move instructions will wait the end of move motion. Delay just wait after previous motion, while the Stable continue to command target location for the given time. It helps the stabilization of robot motion and increase of accuracy.

The non-motion instructions after delay and stable will be executed immediately like Move instructions..

- ApproJ location, dist:
- ApproL location, dist :
- DepartJ dist
- DepartL dist

In many cases you will want to approach a location from distance offset along the tool Z axis or depart from a location along the tool Z axis before moving to the next location.
With approach instructions you can move the robot to the suitable location offset along the tool Z axis.

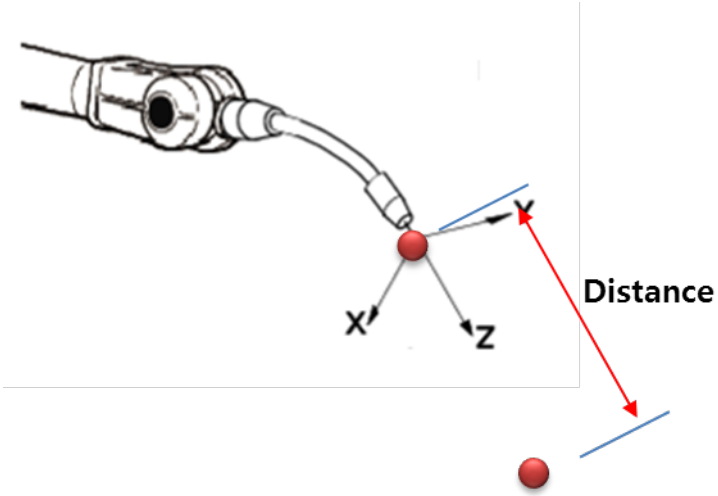


Figure 5 Appro Instructions

- HOME / HOME2 : Two locations can be registered as a home position. To go to the pre-defined location, simply use this instruction. The interpolation mode is joint.

- ALIGN : Align the robot tool Z axis with the nearest world axis.
- Additional Move Arguments : MoveJ p1, [bundle signal no], [speed]
  - Bundle signal no : The output signal timing can be controlled while robot is moving. The conditions are registered as table. This table index is used signal parameters.
  - Speed : Every motion instructions have speed parameters. If speed parameter is only required, you can ignore bundle signal as passing 0 or -1.

- Increment Movement

- IncJ : Move each Axis

  IncJ delta_joint1, delta_joint2, ···

- IncL : Incremental linear Move

  IncL delta_x, delta_y, delta_z ···

- IncT : Incremental Move w.r.t the Tool Axis

  IncT delta_tx, delta_ty, ···

- Drive : The individual joint is moved

  Drive axis_num, delta

```
Drive joint#, value
IncJ -20, 30
IncL 200, 100, 50
IncT 20, 10, -10
```

8.3. Robot Setting

- Robot configuration : These instructions specify the robot configuration. Robot can have multiple postures for the same trans location. Some robots may have no meaningful due to their restricted working range or under constrained kinematics. For instance, SCARA robot have only Lefty/Righty and Cartesian robots have no configurations.
  - ABOVE
  - BELOW
  - LEFTY
  - RIGHTY
  - UWRIST
  - DWRIST
  - SINGLE
  - DOUBLE

- Motion Setting : These instructions specify the robot dynamic properties, such as speed and acceleration.
  - Speed value [Fixed] :

    Determine the speed of robot. If Fixed argument is attached, every motion instructions have this speed. If not used, only the speed of next motion instruction is changed.

```
MoveJ #p1   ; Maximum speed 100%
Speed 20    ;   Next motion speed will be 20% of Maximum.
MoveJ #p2   ; 20% Speed
MoveJ #p3   ; 100% Speed again.

MoveJ #p1   ; Maximum speed 100%
Speed 20 Fixed   ;   For all next motions speed will be 20% of maximum.
MoveJ #p2   ;   20% speed
MoveJ #p3   ; 20% speed, too.

Speed 20 mm/s ; Absolute speed setting , Speed is 20 mm/sec
```

- Accuracy range [Fixed]

  In case robot moves p1→P2→P3 continuously, the robot begins moving toward p2 from p1 by accelerating and it does not decelerate on moving toward p3. Instead, it will smoothly change its direction and begin moving toward P3. It can be defined how smoothly the robot moves at the corner P2.

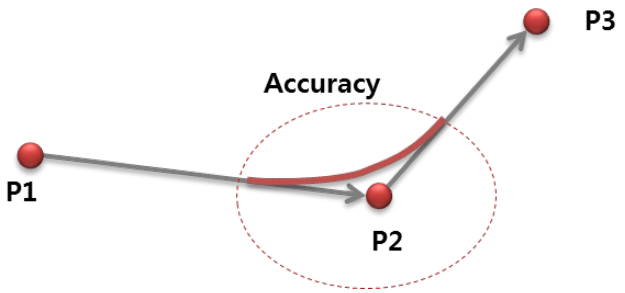


Figure 6 Accuracy 모션

- Accel / Decel accelration% [Fixed]

  The heavy load on the robot causes the unstable motion. This kind of problem can be solved by reducing the acceleration. The maximum acceleration is specified in the robot configurations. The acceleration value is percentage of the maximum acceleration. The range of acceleration is from 0.01% to 100%.

```
Accel 50 : 50% of Maximum acceleration. Applied to the only next motion.

Accel 50 Fixed : Changed for all remained motion
```

- Break : Robot wait until it reaches the exact target positions. It breaks continuous path motion.

```
;prog1
    MoveL p1
    MoveL p2
    MoveL p3
;prog2
    MoveL p1
    MoveL p2
    Break
    MoveL p3
```

As shown above example, if the break instruction is inserted between p2 and p3, MoveL p3 does not start until the robot reaches p2. It breaks the continuous motion.

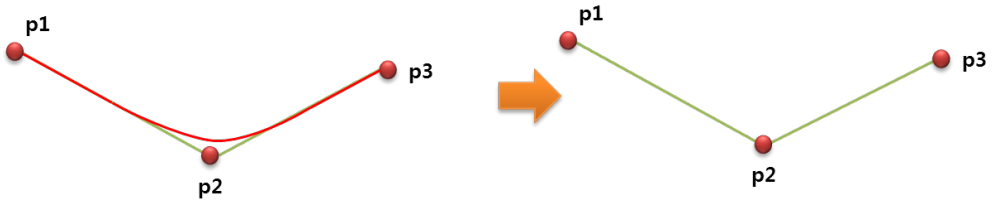


Figure 7 Path changes due to Break instruction

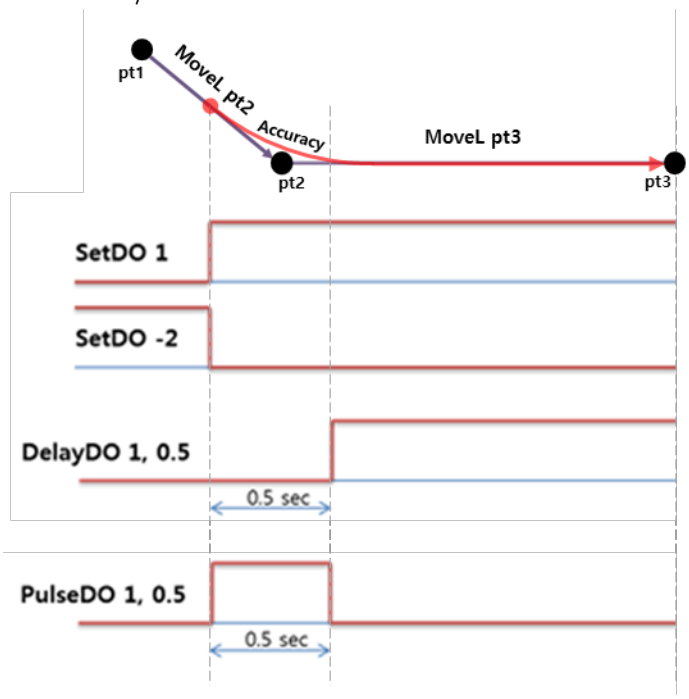
- Brake : This instruction stops the robot motion and then resume the motion to the next

8.4. I/O

- Reset : Reset all digital out signal. All out signal get to off.
- SetDO digital_output_signal_#, ··· : Setting the digital output signal
  - SetDO -sigval, 4 : Signal defined by sigval variable is OFF, Signal 4 is ON

    The negative value turns off the signal and the positive one turns it on.
- PulseDO signal#, time : Pulse output signal. The signal is on during the given time.
- DelayDO signal#, time : Delay outut signal. The signal is on after the given time.
- RunMask startsignal#, count : Masking the signals. From start signal number to the amount of count, the signal will reset when the program stops execution. The default action preserves the signal state even though the program execution is terminated.
- Bits start, count = value : The signal is set as a combined value.
  - Bits 1,8 = 255 : The 8 port signal changes together like a integer variable.
- WaitSig signal#, ... : Wait until the signal conditions meet.

The figure below shows the change of signal line when SetDO, DelayDO, PulseDO is performed after MoveL pt2 command is executed.IO operation position is the beginning of Accuracy.



In addition to this signal processing, when you want to adjust the signal timing by 5mm before arrival, 0.5 second before arrival, or 5mm before departure 5mm, you can apply Msig function setting by complicated setting like signal output.

8.5. Touch Probe

- TouchEnable
  - TouchEnable TRUE 1 : Enable the touch function.
  - TouchEnable TRUE 3, 4 : Activate each channel by different type.
  - ex) TouchEnable TRUE, 1
  - signal_type
    1 : Rising then Falling
    2 : Falling then rising
    3 : Rising
    4 : Falling

- TouchAxis axis1, axis2, axis3, ...
  - Monitoring aixs flag
  - ex) TouchAxis 1,1,1 <- 1,2,3 axis monitoring

- TouchStart probe1, probe2
  - Monitoring start, probe1, probe2
  - ex) TouchStart 1, 1 <- 1, 2 axis probe monitoring start
- TouchStop : monitoring stop
- TouchWait timeout, result
  - Result wait : Wait until the touch point is detected
- TouchDataCount(axis_index, probe_index): Return the detected probe count.
- Detected Touch position
  - ndata = TouchDataCount(probe_index)
  - #TouchJoint(Probe_index, Count_index)
  - TouchTrans(Probe_index, Count_index)
- TouchPerAxis On/Off : each aixs data monitoring mode On/Off
  - On: Acquire axis position data individually as each axis is detected by the probe.
  - Off: Acquire axis position data collectively when all axis is detected by probe.
- TouchReadSDO On/Off : Turn on / off the function of TouchRead which reads data through SDO
- TouchRead : Acquire Touch Data with SDO. Start using TouchRead to acquire Touch Probe Data with SDO.

8.5.1. Example1 : Change Target point 1

```
TouchEnable TRUE 1 ; Touch Enable and 1 : RTOF, 2 : FTOR, 3 : R, 4 : F
TouchAxis 1, 0, 1, 0 ; Set axis to monitor
TouchStart 1, 1 ; probe 1 and probe 2 start monitoring
MoveL ptarget ; move the robot as a ptarget.
TouchWait 2, result ; Switch to next sentence when touch start condition is
established
TouchStop ; Stop touch monitoring
Brake ; Motion stop
; Read the acquired data.
npoint1 = TouchDataCount(1) ; touched point size
npoint2 = TouchDataCount(2) ; touched point size
IF result THEN
Point #ptouch1r = #TouchJoint(1, 1) ; RTOF   -> 1 R, 2 : F
Point #ptouch2r = #TouchJoint(2, 1) ;
Point #ptouch1f = #TouchJoint(1, 2)
Point #ptouch2f = #TouchJoint(2, 2)
END
; calc new target from touch locations : ptarget2
MoveL ptarget2
```

8.5.2. Example2 : Target point scanning → Must be use the TouchWait. function

```
TouchEnable TRUE, 3 ; Touch Enable , Rising edge detect
TouchAxis 1, 0, 1, 0 ; Set axis to monitor : 1, 3 axis only i/o connected
TouchStart 1, 0 ; probe 1 only start monitoring
MoveL ptarget ;
Break ; ptarget motion wait
; Read the acquired data.
nloc = TouchDataCount(1) ; touched point size
for I = 1 to nloc ; rising, rising,  ···
Point tp[i] = TouchTrans(1, i)
```

8.5.3. Example3 : Change Target point 2

```
.LDef shiftmm = -200
Point #pp1 = Joint(0,30,30,0,30,0)
Point pstart = CvtTrans(#pp1)
Point ptarget = Translate(pstart,0,1000,0)
Point ptarget2 = Translate(ptarget,shiftmm,0,0)
MoveJ   Joint(0)
MoveJ   #pp1
WaitTime 2
Speed 100 Fixed
MoveL   ptarget
WaitSig (1008) ; WaitSig operation proceeds while moving to pratget position.
;calc
shiftmm = shiftmm-100 ; The calculation proceeds when a signal is received at 8
on WaitSig.
Point ptarget2 = Translate(ptarget,shiftmm,0,0)
ChgDest ptarget2 ; Change the motion in the ptarget position to the ptarget2
position.
```

8.5.4. Example4 : Individual acquisition Axis Data

```
naixs = 8
Point #p10 = Joint(50,50,50,50,50,50,50,50)
Point #p1 = Joint(0,0,0,0,0,0,0,0)
TouchPerAxis ON
TouchEnable   TRUE, 1
TouchAxis 1,1,1,1,1,1,1,1
MoveJ   #p10
TouchStart   1, 1
MoveJ   #p1
TouchWait   10, result
TouchStop
IF result THEN
    FOR i = 1 TO naixs
        p1_axis[i] = TouchAxisCount(i,1)
        p2_axis[i] = TouchAxisCount(i,2)
        TPWrite 2,"P1_%d = %d, P2_%d = %d",i,p1_axis[i],i,p2_axis[i]
    END
END
WaitTime 1

FOR i = 1 TO 2
    Point #tp_joint1[i] = #TouchJoint(1,i)    ; get probe1 data
    Point #tp_joint2[i] = #TouchJoint(2,i)    ; get probe2 data
END
```

```
FOR i = 1 TO naixs
    prb1_joint_r[i] = GetJ(#tp_joint1[1], i)
    prb1_joint_f[i] = GetJ(#tp_joint1[2], i)
    prb2_joint_r[i] = GetJ(#tp_joint2[1], i)
    prb2_joint_f[i] = GetJ(#tp_joint2[2], i)
END

TouchPerAxis OFF
```

8.5.5. Example5 : Read the Data through the SDO

```
TouchReadSDO On ; Option enable
TouchEnable TRUE 1 ; Touch Enable and 1 : RTOF, 2 : FTOR, 3 : R, 4 : F
TouchAxis 1, 0, 1, 0 ; Set axis to monitor
TouchStart 1, 1 ; probe 1 and probe 2 start monitoring
MoveL ptarget ; Run the robot as a target.
TouchWait 2, result ; Switch to next sentence when touch start condition is
established
TouchRead ; Data read through SDO
TouchStop ; Stop Touch monitoring
Brake ; Motion stop
;
npoint1 = TouchDataCount(1) ; touched point size
npoint2 = TouchDataCount(2) ; touched point size
IF result THEN
Point #ptouch1r = #TouchJoint(1, 1) ; RTOF -> 1 R, 2 : F
Point #ptouch2r = #TouchJoint(2, 1) ;
Point #ptouch1f = #TouchJoint(1, 2)
Point #ptouch2f = #TouchJoint(2, 2)
END
; calc new target from touch locations : ptarget2
MoveL ptarget2
```

8.6. Network Command

8.6.1. Client Mode

- TCPConnect sockets variable, IP address as string, port_number
- TCPClose socket variable
- TCPRead socket variable, string variable, [return code]
- TCPWrite socket variable, string_variable, [return code]

8.6.2. Server Mode

- TCPSStart socket_variable, port number

- If socket variable is negative, the error has occurred.

- TCPSStop soket_variable
  - Close the socket.
- TCPSAccept soket_variable, client_socket_variable
  - If Client_socket_variable is negative, the error has occurred.
- TCPSCClose client_socket_variable
  - Close the Client_socket.
- TCPSRead client_socket, string variable, [return code]
  - Read the date from client.
- TCPSWrite client_socket, string variable, [return code]
  - Write the date to client.

In example section of this manual, the vision interface application shows how to use network instructions. As the communicated data type is string, it is necessary to change to a right data type , such as numeric. Before converting to numeric, the received string must split as a token list. SplitStr instructions make the string to split as the tokens. Then, each token can be converted to numeric with Value instruction.

<SplitStr instruction example>

```
"%-2.1,15.4,95" is converted to   tx = -2.1, ty = 15.4, trot = 95 as follows.

  $msgread = "%-2.1,15.4,95"        ; received string data

SplitStr $token[0], $msgread, "%," ; split string

      ; '%', ',' are separator characters to split string

      ; The result will be $token[0] = "-2.1", $token[1] = "15.4", $token[2] = "95"

Tx = Value($token[0])      ; convert to numeric

Ty = Value($token[1])

Trot = Value($token[2]
```

8.7. Conveyor Tracking

- TkSetSig cvid, trigger signal #
  - Specify the trigger signal to add data to the object queue..
  - You don't need to use it in the program because you can also define it in the Conveyor settings.
  - In the Conveyor configuration, you can substitute the trigger signal through the latch state of the counter.
- TkObjAdd cvid: In the case of a vision system that recognizes multiple objects at once, one object is added at first recognition, so if you add a new object, you can analyze and add vision data.
- TkObjClear cvid: remove the Object List.
- TkObjDropCur cvid: Drop object from current position.
- TkObjGetType: Obtains the registered Type when the Object is added. For a vision system that recognizes objects with multiple shapes, it is necessary to input not only the conversion position but also the type to distinguish the shape, so it is used to distinguish the moving position according to the input type.
- TkFilterNullShift: It is a function to filter (delete) when data which is not big difference with existing object comes in. Duplicate data may be detect when using vision system and delete it.
- TkObjWait cvid, timeout, result : Waits for objects on the conveyor to enter the workable area. If there is no Timeout value, it is an infinite wait. If there is a value, it waits for a given time. If Object is selected at this time, result = TRUE and object selection fails, result becomes false.
- TkMove cvid, pt : Moves linear interpolated motion tracking the conveyor.
- TkAppro cvid, dist : Moves the distance along the toolz direction from previous motion target location.
- TkDepart cvid, dist : Moves the reverse direction along the toolz from current location.
- TkMoveC cvid, p1, p2 : Circular interpolated motion synchronized the conveyor. It looks like an elliptic shape from the view of the outside.
- TkStop cvid : Stops the conveyor tracking.
- TkObjGet cvid, tx, ty, trot : Gets the information for the last inserted object.
- TkObjShift cvid, tx, ty, [trot=0] [tz=0] : Sets the shift information for the last queued object. Normally the shift information is transferred form the vision system. The third argument is rotation information not z translation because it is rarely used.
- TkObjRemove cvid : Removes the last queued object.
- TkObjClone cvid, [result]

- The last queued object is copied and queued. It has the same conveyor position, but no shift information.
- Result == TRUE is success in cloning.

- TkObjFilterEnv cvid, enable_flag, check_radious, search_count
  - Sets the object filter
  - Check_radius is distance criteria determining the same object
  - Search count is the number of existing objects to test filtering
- TkObjFilter cvid
  - Do a filter test for the last queued object. If the object is the same object to the existing objects, it removes from the object queue.
- Nobj = TkObjCount(cvid)
    - Returns the number of objects. With this functions you can also decide the object existence.

<Conveyor Tracking Example : MainTask >

```
; move home position
dist = 70                ; approach distance
convid = 1               ; conveyor id
Accuracy 50 Fixed        ;
Point phome = Trans(20,0,680)
Point pdrop = Trans(190,0,675)
Point ppick = Trans(0)          ; Origin location w.r.t the object coordinate
MoveL phome                     ; Move to wait position
TkObjWait convid                ; Wait until object is valid
; catch target - ppick is relative w.r.t the conveyor reference
TkMove convid,ppick
TkAppro convid,dist             ; Move to the direction tool-z from ppick
Signal 1,2                      ;
TkDepart convid,dist            ;
Break                           ; Break the continous path motion
MoveL pdrop                     ; Move to drop position
Signal -1,-2
```

- Alternatives instead of TkAppro/TkDepart
  - Point ppick2 = Translate(ppick, 0, 0, 70)
  - TkMove cnvid, ppick2
  - TkMove cnvid, ppick
- If you run this program, the robot move the default target location whenever a trigger signal is received. To identify the actual location with a vision system, the next example can be used together.
- **When resuming the program after stop the middle of the program, it may be required to be Reset and resume at the beginning of the program because it may loose the tracking object information.**

If you need to change the position of the object to be connected to the vision device, you can execute the program that changes the position of the object using the vision information as shown below in SubTask.

<Vision interface example: SubTask >

```
cognex = 0
server = 0
count = 0
vis_ret = 0
; open new socket
TCPSStart server,3240
TCPSAccept server,cognex
WHILE TRUE DO
   $msgread = ""
     TCPSRead cognex,$msgread,vis_ret
     IF vis_ret == 0 THEN
          TPWrite 2,"Vision disconnected",0
          GOTO 100
   END
   SplitStr $token[0],$msgread,"%, "     ; Vision sends dx, dy like this: "%8.3,-2.5"
   ; apply transform
   tx = Value($token[0])
   ty = Value($token[1])
   TkObjShift 1,tx,ty,0
END ; end of while
100       TCPSStop server
```

- In this example, the vision sends only one object deviation by the trigger signal. If you have multiple object, you need more additional instructions such s Filter , Clon
- On stopping in the middle of the program, reset and resume program is required because TCP connection is lost.
- Only to test the communication with the vision system, you can check the result using TPWrite instruction

```
TCPSRead cognex, $msgread, vis_ret
TPWrite 2, "Vision : %s", $msgread
```

- In real situation, the diagnostic message consumes the CPU and the system performance can be degraded. Therefore it may be used for only test the system.

8.8. ETC

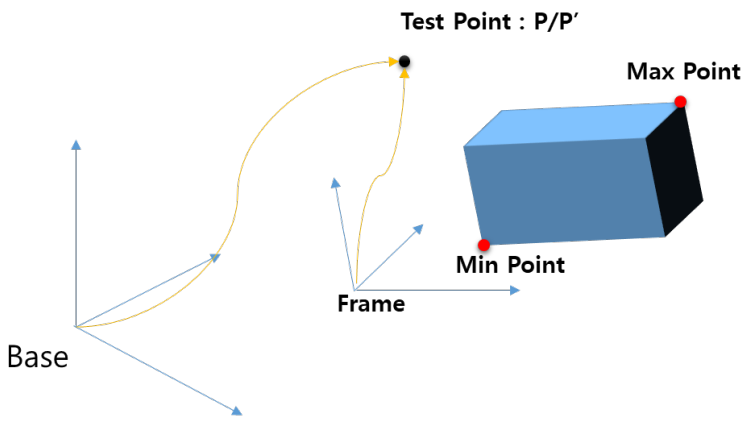
- ULIMIT / LLIMIT : Set the upper limit, lower limit
- TIMER

Timer setting : SetTimer timer#, time
Timer read : Timer(timer#)

```
SetTimer 1, 0
MoveJ #p1
SetDO 1, -2
Cycletime = Timer(1)
TPWrite ICONI, "Cycle time = %f", cycletime
```

8.9. World Zone

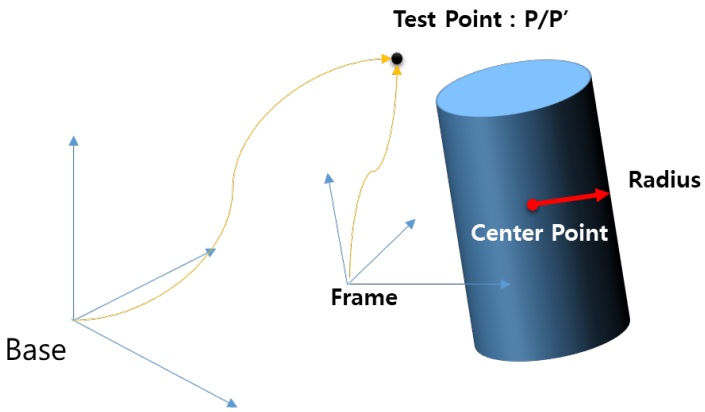
- Cartesian Limit Zone
- Apart from the limit setting of the joint area, set the inner area by combination of Box and Cylinder for the base coordinate system.
- WZBox: Set Box area.



- WZBox id, operation, p1, p2, [frame]
  - Id: worldzone id
  - Operation:

    1: Add
    -1 : Subtraction
  - P1: min point (x, y, z)
  - P2: max point(x, y, z)
  - Frame: frame of WZone (default is same as Base coordinate)
- WZBox example:

  WZBox 1, 1, Trans(10, 10, 0), Trans(1000, 500, 500), [Trans(Frame)]
- WZCheck On/Off: On/Off World Zone Checking.
- WZCyl: Set Cylinder area.



- WZCyl id, operation, cp, r, [frame]
  - Id: worldzone id
  - Operation:

    1: Add
    -1 : Subtraction
  - cp: Center point(x, y, z)
  - r: radius
  - Frame: frame of WZone (default is same as Base coordinate)
- WZBox example:

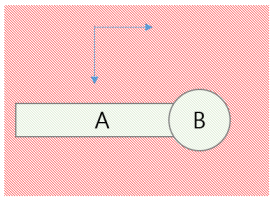
  WZCyl 1, 1, Trans(500, 500, 0), 100, [Trans(Frame)]
  The height of the cylinder is infinity.

- WZClear id: Clear WZone
- WZOff id: Disable WZone [id]

- WZOn id, Region(Selected/Opposite): Activate WZone [id]
  - Region : True/False

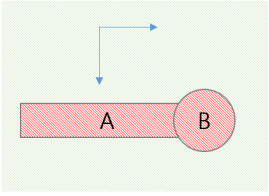
    -When negate is True, all areas become operable. If operation is 1 (Add), an inoperable area is added. If Operation is -1 (Subtraction), an operable area is added.
    -When negate is False, all areas become inoperable area. If operation is 1 (Add), an operable area is added. If Operation is -1 (Subtraction), an inoperable area is added.
- WZSignal signal: When TCP comes in WZ, signal activated.
- Composite Zone: Add, Subtraction

Zone 1 : A+B



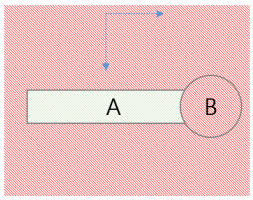

```
;Frame:
Point wz_frame = Trans(0,0,0)
;A:
Point box_min = Trans(0,0,0)
Point box_max = Trans(10,100,10)
;B:
Point cyl_center = Trans(0,0,0)
radius = 15

WZClear 1
WZBox 1, 1, box_min, box_max, wz_frame
WZCyl 1, 1,cyl_center, radius, wz_frame

WZOn 1, FALSE
WZCheck On
```

Zone 1 : NOT(A+B)



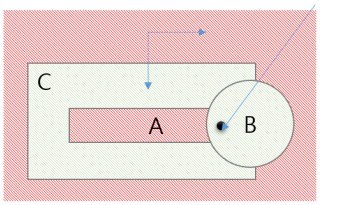

```
;Frame:
Point wz_frame = Trans(0,0,0)
;A:
Point box_min = Trans(0,0,0)
Point box_max = Trans(10,100,10)
;B:
Point cyl_center = Trans(0,0,0)
radius = 15

WZClear 1
WZBox 1, 1, box_min, box_max, wz_frame
WZCyl 1, 1, cyl_center, radius, wz_frame

WZOn 1, TRUE
WZCheck On
```

Zone 2 : A-B




```
;Frame:
Point wz_frame = Trans(0,0,0)
;A:
Point box_min = Trans(0,0,0)
Point box_max = Trans(10,100,10)
;B:
Point cyl_center = Trans(0,0,0)
radius = 15

WZClear 2
WZBox 2, 1, box_min, box_max, wz_frame
WZCyl 2, -1, cyl_center, radius, wz_frame

WZOn 2, FALSE
WZCheck On
```

Zone 3 : C – A + B

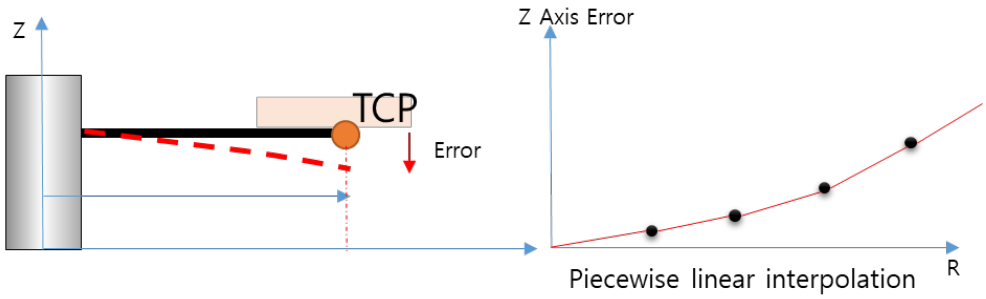



```
;Frame:
Point wz_frame = Trans(0,0,0)
;A:
Point box_min = Trans(0,0,0)
Point box_max = Trans(10,100,10)
;B:
Point cyl_center = Trans(0,0,0)
radius = 15
;C:
Point box2_min = Trans(0,0,0)
Point box2_max = Trans(100,100,10)

WZClear 3
WZBox 2, 1, box2_min, box2_max, wz_frame
WZBox 2, -1, box_min, box_max, wz_frame
WZCyl 2, 1, cyl_center, radius, wz_frame

WZOn 3, FALSE
WZCheck On
```

Unreachable Zone
Robot Working Zone

8.10. ZComp

Store Z-axis error as a table and the deflection of Z axis is compensated in programmed motion.



- R: The vertical distance of Z axis at TCP position.
- The direction of the Z-axis and the direction of the Error must be reversed.
- Error Table

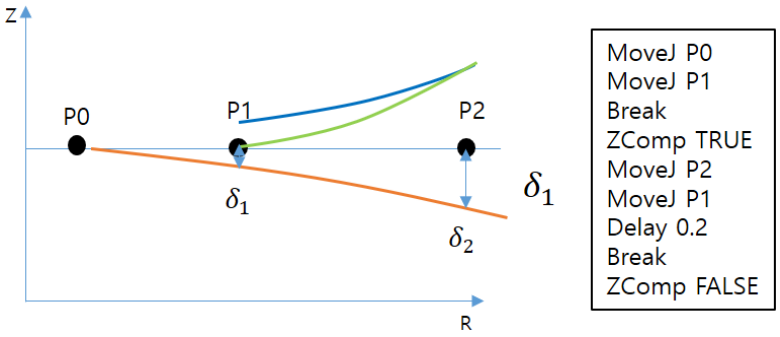
R	Error
500	0.5
600	0.65
800	0.8
1000	1

  - File location: /mnt/mtd5/zcomp.ini
  - File format

```
5          → item number
0, 0       → first R, Z data
150, 0
300, 5
500, 10
600, 20
```

- ZComp TRUE/FALSE: On/Off ZComp.
- Precautions when using ZComp

1) If enable / disable is enabled in the section where the error exists (P1 ~ P2), the reciprocating trajectory may be changed.



As shown in the figure above, when there is an error of δ_1 in P1. when moving from P1 to P2, the operation plan is (P1- δ_1) -> P2.

At this time, in order to go to P2, it actually moves to the position of (P2 + δ_2).

When moving from P2 to P1, it moves to the position of (P1 + δ_1Therefore, if Enable / Disable in the section where the error exists, the reciprocating trajectory may be changed.

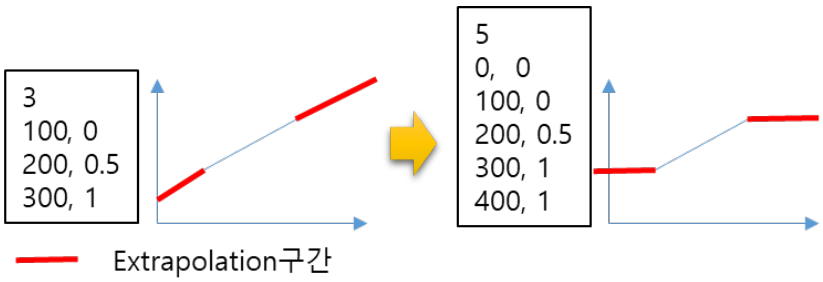
2) Enable / Disable during operation

Since the ZComp status (Error correction status / Error non-compensation status) can be changed, So change it after completing the operation using the Break command

3) Extrapolation

Extrapolation is performed in an interval other than the error table setting, so if you want to leave a constant outside the error definition interval, you should add a point that does not change the error.

.



8.11. User Coordinate Instruction

If a custom robot does not have the Cartesian coordinate but the special coordinate called a user coordinate, the linear interpolated motion with respect to the user coordinate is provided in CL. For the multi-hand TCP the undetermined motion can be resolved by setting master slave relationships.

- UCMove #ptarget : Linear interpolated motion on User coordinate

- UCMasterArm arm1, [arm2], [arm3], [arm4] : Specify the master slave order
    - If the user coordinates have two XYZ sets by 2 Arms. The arm motion is calculated first set as a master arm, ant the other's follows the first.
- UCHint $hint_name, value : Sets the options for Custom ARM. It can be any types of parameter value set.

.

9. Diagnostic Functions

- TPWrite icon, str_format, [arg1], [arg2], [arg3], …, [arg10]
  Print the message on the message window. The format of string is the same as the c standard function, printf. But, the number of argument can be up to 10. The level of message is specified by icon as well as displaying icon. For error icon the message is also saved to system log.
  - icon = 0 : error 1 : warning, 2 : information
    Messages corresponding to the error will be recorded in the system log.
  - CL has predefined constant for icon. ICONE = 0, ICONW = 1, ICONI = 2.
    - TPWrite ICONI, "message"
  - cycle = cycle + 1
  - TPWrite 2, "program cycle = %d", cycle
- TPClear : Clear the message out area..
- RaiseError user_erro_code :
  When the system error occurred, it displays in the message area and is saved to log. Sometimes the robot program stops due to the error. CL provides how to define this kind of error behavior as a user defined error. Application programmer defines the application specific error code and raises it in programs.

  The error is raised, the robot stops and the message is logged.

  User error code can be used from -9000 to -10000.
  - ex) RaiseError -9001
- How to Auto Restart at Error Stop
  - Detect Error output signal in SubTask.
  - Send Error reset from SubTask.
- AbortOnUserErr
  - When User Error occurs in Subtask, it determines whether Subtask is stopped or not.
  - Subtask non-stop and Maintask stop when user error occurs in Subtask.
  - If AbortOnUserErr is set to On, Subtask stops when a user error occurs in Subtask.
- AutoSleep: On / Off control 1 cycle sleep in Loop finish.
  - Be careful when using loop statements. WaitTime (or a command that can give a time delay) should be inserted (at least 1 cycle time, 2 msec) in situations where the ending point of a while or GOTO statement is opaque.

10. Program selection with an external i/o

Process controller such as PLC selects the robot program with digital io signal. It can be accomplished using IO instructions of CL. And, CL provides simpler method to implement it.
CL maps the ranges of the digital input to numbered program and chooses the program following the signal values.
To select program externally, EPSMode must be enabled and the program is selected at EPSWait instruction.

1) Macro Command

- EPSmode On or EPSmode Off

  Enable or disable the external program selection mode.

- EPSWait

  Wait signal and jumps selected program.

2) To use EPS the 6 dedicated signals must be defined. It is defined on the robot configuration file.

- Config Example)

- Output;

DDCO_EPS_MODE	ex) DDCO_EPS_MODE = 10, 1
DDCO_EPS_STATUS	ex) DDCO_EPS_STATUS = 11, 1

- Input :

DDCI_EPS_ON	ex) EPS_ON = 1007, 1
DDCI_EPS_OFF	ex) EPS_OFF = 1008, 1
DDCI_EPS_START_BIT	ex) DDCI_EPS_START_BIT = 1009
DDCI_EPS_END_BIT	ex) DDCI_EPS_END_BIT = 1012

In the example, the program number is defined with 4 bit range, 1009~ 1012. Therefore program name can be Pg1, Pg2, .. Pg9, Pg10, Pg11, Pg12, Pg13, Pg14, Pg15.

EPS_START_BIT ~ EPS_END_BIT: IO bits to define program
In case of value 1 ~ 9 : The name of program must be Pg1 ~ Pg9
In case of value 10 ~ 99 : The name of program must be Pg10 ~ Pg99
In another case, 100 ~ 999: The name of program must be Pg100 ~ Pg999

When EPSMode is ON, and runs EPSWait, the EPS_STATUS becomes ON. , The PLC checks the EPS_STATUS and selects the program. After selecting a program

PLC makes the robot runs the program with EPS_ON or cancels the progam EPS_OFF.

10.2. Example : Program selection with EPS MODE

- Macro Example EPS Macro (EPS Setting Macro Program name: EPSGO)

```
HOME
EPSMode ON
EPSWait
```

- Macro Example pg1 (Macro Program name : Pg1)

```
MoveJ #p1
MoveJ #p2
```

- Macro Example pg2 (Macro Program name: Pg2)

```
MoveJ #p3
MoveJ #p4
```

- Operate the .EPSGO program
- EPSMode turns ON.
- Wait the EPS On signal in EPSWait step.
- Set the signal 1012 = off, 1011 = off, 1010 = off, 1009 = on
- - EPS_ON is on, then Pg1 runs
- - Set the signal 1012 = off, 1011 = off, 1010 = on, 1009 = off
- - EPS_ON is on, then Pg2 runs

10.3. Program selection with Bits instruction and SWITCH CASE structure

```
WaitSig 1007
Callprog = Bits(1009,4)
SWITCH callprog
CASE 1
Call myprog1
CASE 2
Call myprog2
```

10.4. calling a program using EPSWait conditional statement

- Macro Example EPS Macro (EPS Setting Macro Program name: EPSGO)

```
Result = 0
EPSMode ON          ;EPSMode ON.
EPSWait 2,result    ;result = true if there is no EPS signal for 2 seconds
IF result == TRUE THEN
Call other()        ;call other()
END
```

- Macro Example pg1 (Macro Program name : Pg1)

```
MoveJ #p1
MoveJ #p2
```

- Macro Example pg2 (Macro Program name: Pg2)

```
MoveJ #p3
MoveJ #p4
```

- Macro Example other (Macro Program name: other)

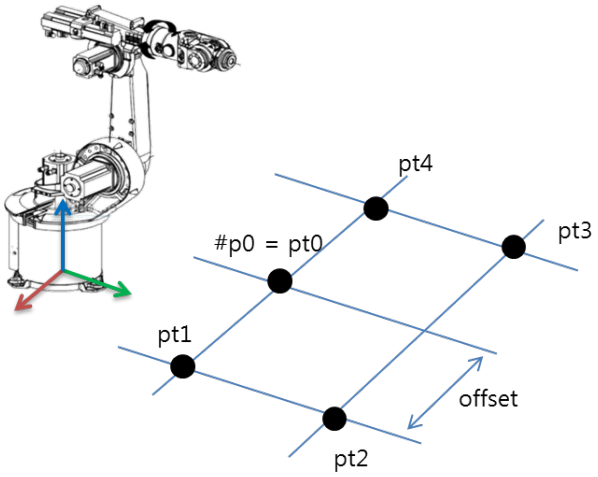
```
MoveJ #p5
MoveJ #p6
```

- Explanation of the example
  - Run the .EPSGO program
  - Enable the eps mode
  - Wait the EPS on signal in EPSWait step.
  - Macro Signal input
    1012 = off, 1011 = off, 1010 = off, 1009 = on
  - Enter EPS_ON within 2 seconds to execute Pg1.
  - Macro Signa input
    1012 = off, 1011 = off, 1010 = on, 1009 = off.
  - Enter EPS_ON within 2 seconds to execute Pg2.
  - If time passed 2second, than the other() program will be start.

11. Example programs

11.1. Basic motion program

#p0 : A teaching location .
Offset : Predefined or programmed numeric variable
Robot moves #p0 □ pt1 □ pt2 □ pt3 □ pt4 □ pt0.




```
MoveJ #p0
Offset = 100          ; offset value
Point pt0 = CvtTrans(#p0)        ;   convert Joint variable to Trans variable
Point pt1 = Translate(pt0, offset, 0, 0)
Point pt2 = Translate(pt1, 0, offset, 0)
Point pt3 = Translate(pt2, -2*offset, 0)
Point pt4 = Translate(pt3, 0, -offset, 0)
MoveJ #pt0
MoveL pt1
MoveL pt2
MoveL pt3
MoveL pt4
MoveL pt0
```

11.2. CP Motion

By increasing the accuracy at Pt2, the motion at the corner becomes more continuous. If the load is heavy, it helps more stable motion by decreasing the acceleration.

```
MoveJ #p0.
Offset = 100
Point pt0 = CvtTrans(#p0)
Point pt1 = Translate(pt0, offset, 0, 0)
Point pt2 = Translate(pt1, 0, offset, 0)
Point pt3 = Translate(pt2, -2*offset, 0)
Point pt4 = Translate(pt3, 0, -offset, 0)
MoveJ #pt0
MoveL pt1
Accuracy 50
Accel 50
Decel 50
MoveL pt2 ; Accuracy/Accel/Decel affects only to move pt2
MoveL pt3 ; The motion settings is recovered on moving pt3MoveL pt4
MoveL pt0
```

- To change the settings permanently, use like this : Accuracy 50 Fixed, Accel 50 Fixed, Decel 50 Fixed. The motion parameters are fixed, they are effective for remained motion instructions.

11.3. IO instruction

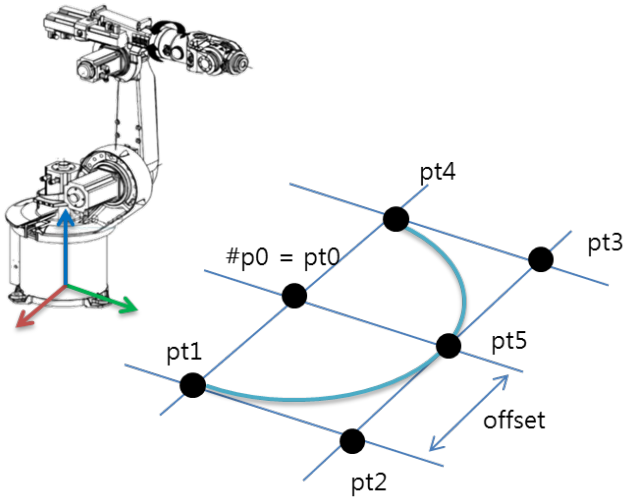
This examples shows how to use Digital Input and Digital output signal.

```
MoveJ #p0
Offset = 100
Point pt0 = CvtTrans(#p0)
Point pt1 = Translate(pt0, offset, 0, 0)
Point pt2 = Translate(pt1, 0, offset, 0)
Point pt3 = Translate(pt2, -2*offset, 0)
Point pt4 = Translate(pt3, 0, -offset, 0)
MoveJ #pt0
WaitSig    1002    ; Wait until Digital Input 1002 is on
MoveL pt1
Accuracy 50
Accel 50
Decel 50
MoveL pt2
SetDO 3, 5; When start to move to position pt3, Set the signal 3 On.
MoveL pt3
SetDO -3, -5        ; off the output signal 3 and 5.
MoveL pt4
MoveL pt0
```

- Delayed output or Pulse output is needed, use the instruction, DelayDO or PulseDO.
- With Bundle IO, the output timing can be controlled during a motion.

11.4. MoveC

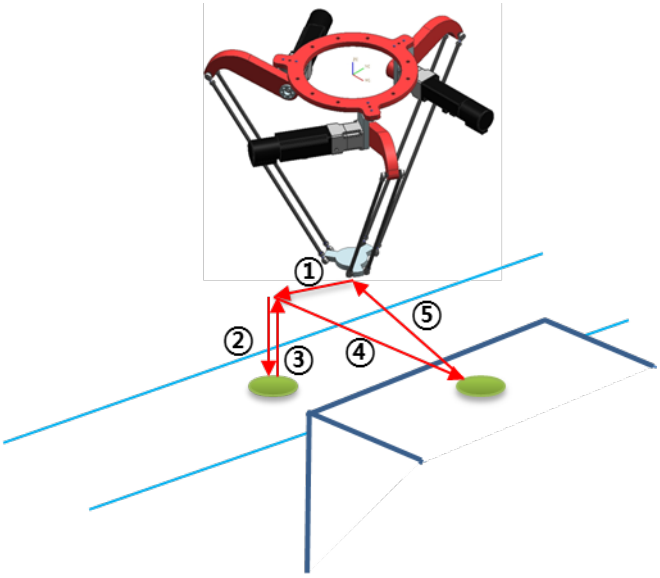
This example shows how to move robot following a circular path. To define a circular movement, two locations must be defined.



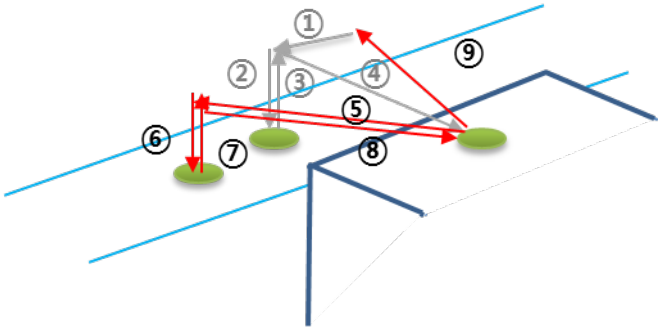

```
MoveJ #p0
Offset = 100
Point pt0 = CvtTrans(#p0)
Point pt1 = Translate(pt0, offset, 0, 0)
Point pt2 = Translate(pt1, 0, offset, 0)
Point pt3 = Translate(pt2, -2*offset, 0)
Point pt4 = Translate(pt3, 0, -offset, 0)
Point pt5 = Translate(pt0, 0, offset, 0)
MoveJ #pt0
MoveL pt1
MoveC pt5, pt4        ; Two locations are required.
MoveL pt0
```

11.5. Conveyor Tracking and Pickup

The order of robot movement is like the figures, but the robot does not move to the wait location when multiple workpieces exist on the conveyor to reduce cycle time.



No waiting other workpiece: move 1→2→3→4→5



Multiple workpieces exist : Robot skips going to 5 and goes to the next workpiece directly.

```
dist = 50
convid = 1
Point phome = Trans(0,0,480)
Point pdrop = Trans(200,-200,480)
Point pdrop2 = Translate(pdrop,0,0,dist)
        Point ppick = Trans(0)+Rz(-90)
        Accuracy 100 Fixed
100      MoveL   phome
        wait_result = FALSE
200      Break
        TkObjWait convid,1,wait_result
        IF wait_result == FALSE THEN
            GOTO 100   ; No workpiece, goes to waiting location
        END
```

```
        Accuracy 5 Fixed
        TkMove convid,ppick
        Accuracy 0.5
        TkAppro convid,dist
        TkDepart convid,dist
        MoveL   pdrop
        Accuracy 0.5
        MoveL   pdrop2
        MoveL   pdrop
;
        objcount = TkObjCount(convid)
        IF objcount>0 THEN   ; There exists object, goes to workpiece location
            GOTO 200
        END
```

Break instruction is required in between the tracking motion and non-tracking motion to distinguish them.

11.6. Base Move / Tool Move

```
Example of move by Base or Tool
Point #p1 = Joint(0, 0, 0, 0, 32, 0)
Point tans_pos = trans(321, 0, 500, 0, 132, 0)
MoveJ   #p1

;//base move (0,0,20)
MoveL   tans_pos
Point tool_co_move = translate(tans_pos,0, 0, 20)
MoveL tool_co_move

;//tool move (0, 0, 20)
MoveL   tans_pos
Point base_co_move = tans_pos + trans(0, 0, 20)
MoveL base_co_move
```

12. Function Reference

Explanation of function commands to help calculation and various operations of Macro Program

12.1. Abs – Return absolute value.

Description:

The argument value is converted to an absolute value.

Syntax:

value1 = Abs(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number.

Example:

a1= -30

a2 = 10

dist = Abs(a1 – a2).

Result:

dist : 40

12.2. ACos – Returns the arccosine value.

Description:

Calculate and return the arc cosine value of the input value.

Syntax:

angle = ACos(value)

Return value: angle

Data type : Number

Arguments: value

Data type : Number

Example1:

acosdg1= ACos(0)

Result:

acosdg1 = 90

12.3. AIn – Returns the arccosine value.

Description:

Calculate and return the arc cosine value of the input value.

Syntax:

angle = ACos(value)

Return value: angle

Data type : Number

Arguments: value

Data type : Number

Example1:

acosdg1= ACos(0)

Result:

acosdg1 = 90

12.4. AOut – Returns the output analog value.

Description:

Get analog output value

Syntax:

value1 = AOut(channel)

Return value: value1

Data type : Number

Arguments: channel

Data type : Number(AO channel)

Example:

aochannel2= AOut(2)

12.5. Asc – get ASCII code

Description:

Returns the ascii value corresponding to the String index.

Syntax:

value1 = Asc("CoreRobot", index)

Return value: value1

Data type : Number

Arguments: "CoreRobot", index

Data type : $String, Number

Example:

ascii1 = Asc("CoreRobot", 1)

get ascii value of the first index 'C'.

Result:

ascii1 : 67

12.6. ASin – Return arcsine value.

Description:

Calculate and return the arc sin value of the input value.

Syntax:

angle = ASin(value)

Return value: angle

Data type : Number

Arguments: value

Data type : Number,.

Example1:

asinedg1= ASin(1)

Result:

asinedg1= 90

12.7. Atan2 – return Arc tangent2.

Description:

Calculate and return the arc tan2 value of the input value.

Syntax:

value1 = Atan2(y, x)

Return value: value1

Data type : Number

Arguments: y, x

Data type : Number

Example:

y = 1

x = 1

degree1 = Atan2(y,x)

Result:

degree1 : 45

12.8. Bits – Converts the value of bits to a number.

Description:

Reads a consecutive number of signal values from the start signal number

Syntax:

value1 = Bits(signal, amount)

Return value: value1

Data type : Number

Arguments: signal, amount

Data type : Number

Example1:

cnum = Bits(1009, 4)

Example2:

```
prgnum = Bits(1009, 4)
SWITCH prgnum
CASE 1:
        macro1
CASE 2:
        macro2
DEFAULT:
END
```

DI 1012, 1011, 1010, 1009 s

If 1012 = off, 1011= Off, 1010 = On, 1009 = off, bit = 0010

0010 is converted to decimal number, 2 is assigned to progrum, and CASE 2 is executed.

12.9. CheckSum – Check the validity of communication data

Description:

When transmitting data in serial communication, check whether there is an error in the transmitted data. There are five ways to check data transmission error: CRC16, VRC, LRC (BCC), SUM, OR.

Syntax:

chkvalue = CheckSum($data, type)

Return value: chkvalue

Data type : Number(Dec)

Arguments: $data, type

Data type : String, Number

type:

0 : CRC16

1 : VRC(Vertical Redundancy Check, XOR)

2 : LRC(Longitudinal Redundancy Check, XOR)

3 : SUM(sum)

4 : OR(or)

Example1:

$data = "123"

type = 3

chkvalue = CheckSum($data, type)

Result:

Chkvalue : 150

ASCII char : '1' -> Dec : 49

ASCII char : '2' -> Dec : 50

ASCII char : '3' -> Dec : 51

chkvalue = 49+50+51 = 150

12.10. $Chr – Converts Hex code to Ascii Char.

Description:

Converts Hex code to Ascii Char.

Syntax:

$char = $Chr(hex_code)

Return value: $char

Data type : String

Arguments: hec_code

Data type : Hex code

Example1:

hex_code= ^h31

$char = $Chr(hex_code)

Result:

$char: "1"

12.11. Cos – return Cosine value.

Description:

Calculate and return the cosine value of the input value.

Syntax:

value1 = Cos(value2)

Return value: value1

Data type : number

Arguments: value2

Data type : Number

Example1:

cosdg1= Cos(90)

Result:

cosdg1 : 0

12.12. CvtJoint – Convert Trans position to Joint position.

Description:

Convert Trans position to Joint position.

Syntax:

Point #p1 = #CvtJoint(pt1, [#pinit], [isLefty], [isBelow], [isUWrist])

Return value: #p1

Data type : Joint

Arguments: pt1, [#pinit], [isLefty], [isBelow], [isUWrist]

Pt1: Trans

[#pinit]: base joint position

[isLefty]: LEFFTY/RIGHTY

[isBelow]: BELOW/ABOVE

[isUWrist]: UWRIST/DWRIST

Example1:

Point #p1 = #CvtJoint(pt1) ; default init joint = (0, 0, .., 0)
Convert the Trans position pt1 to Joint position #p1 based on Joint(0,0,…)

Example2:

Point #p2 = #CvtJoint(pt2, #pinit, FALSE, FALSE, FALSE)
Convert the Trans position pt1 to Joint position #p1 based on #pinit.
Because Robot configuration optioins are all FALSE, so robot configurations are set to isRighty, isABOVE, isDWrist.

12.13. CvtTrans – Convert Joint position to Trans position..

Description:

Convert Joint position to Trans position.

Syntax:

Point value1 = CvtTrans(#value2)

Return value: value1

Data type : Trans

Arguments: #value2

Data type : #Joint

Example1:

Point #joint1 = Joint(10)

Point ctrans1 = CvtTrans(#joint1)

12.14. Dest/#Dest – Return current motion's destination point.

Description:

Return current motion's destination point(Joint/Trans).

Syntax:

Point #value1 = #Dest

Point value2 = Dest

Return value: value1, #value2

Data type : #Joint or Trans

Trans position use command Dest.

Example1:

Point dtrans1 = Dest

Point dJoint1 = #Dest

12.15. Distance – Return distance between A and B point.

Description:

Return distance between A and B point.

Syntax:

value1 = Distance(A, B)

Return value: value1

Data type : Numver

Arguments: A, B

Data type : Trans, Trans

Example1:

Point dtran1 = Trans(10)

Point dtans2 = Trans(20)

distrance1 = Distance(dtran1, dtran2)

Result:

distrance1 : 10

12.16. DX, DY, DZ – X or Y or Z in Trans variable.

Description:

Return x,y,z value of trans point p

Syntax:

value1 = DX(transvar)

value2 = DY(transvar)

value3 = DZ(transvar)

Return value: value1, 2, 3

Data type : Numver

Arguments: transvar

Data type : Trans

Example1:

Point dtran1 = Trans(10, 20, 30)

dxt1 = DX(dtran1)

dyt1 = DY(dtran1)

dzt1 = DZ(dtran1)

Result:

dxt1 : 10, dyt1 : 20, dzt1 : 30

12.17. ErrorCode – Retrun User error code.

Description:

Return User Error code

Syntax:

value1 = ErrorCode

Range(-9001~-10000)

Return value: value1

Data type : Number

Example1:

a = ErrorCode

12.18. Frame – Create coordinate frame.

Description:

Create coordinate frame.

Syntax:

Point value_frame = Frame(value_org, value_x, value_xy, value_z)

Return value: value_frame

Data type : Trans

Arguments: value_org, value_x, value_xy, value_z

Data type : Trans

Value_org: The origin of the coordinate system

Value_x: a point on the x axis

Value_xy: a point in x_y plane

Value_z: determine z -axis direction

Example1:

```
Point porg = Trans(0,0,0,0,0,0,0)

Point px = Trans(30,0,0,0,0,0)

Point pxy = Trans(30,30,0,0,0,0)

Point pz = Trans(0,0,30,0,0,0)

Point newcoord= Frame(porg, px, pxy, pz)
```

Result:

newcoord : 0,0,30,0,0,0

12.19. $Fill– Assign a buffer to a string.

Description:

allocate the buffer size of the string.

Syntax:

$string = $Fill("a", size)

Return value: $string

Data type : String

Arguments: "value", size

Data type : $String, Number

Max Size Number: 253

Example1:

num_size = 200

$str_var = $Fill("a", num_size)

12.20. $HexStr – Convert hex code to string

Description:

Convert Char to hex code, and return string value type

Syntax:

$hex_code = $HexStr($str)

Return value: $hex_code

Data type : string

Arguments: $str

Data type : string

Example1:

ASCII char : '1' -> Hex: 31

ASCII char : '2' -> Dec : 32

ASCII char : '3' -> Dec : 33

Program:

$str = "123"

$hex_code = $HexStr($str) ;313233

12.21. Here/#Here – Assign the current location(Joint/Trans).

Description:

Get current position coordinate(Joint/Trans).

Syntax:

Point #value1 = #Here

Point value2 = Here

Return value: #value1 or value2

Data type : #Joint or Trans

Here: get Trans coordinate, #Here: get Joint coordinate.

Example1:

```
Point gotrans1 = Trans(10,10,10)

MoveL gotrans1

Point tphere1 = Here

Point #gojoint1 = Joint(20,20,20)

MoveJ #gojoint1

Point #jphere1 = #Here
```

Result:

tphere1 : 10,10,10

#jphere1 : 20,20,20

12.22. #Joint – Create a Joint position variable with Joint values for each axis.

Description:

Create Joint variable

Syntax:

Point #value1 = Joint(j1, j2, j3, j4, j5, j6)

Return value: #value1

Data type : #Joint

Arguments: j1 ~ 6

Data type : Number

J1~J6: each joint axis.

Example1:

```
n1 = 10

n2 = n1

n3 = n1 + n2

n4 = n1^2

n5 = n1 * n4

n6 = 20

Point #joint1 = Joint(n1,n2,n3,n4,n5,n6)
```

Result:

#joint : 10, 10, 20, 100, 1000, 20

12.23. JVal – Get a specific joint value of axis.

Description:

Get Joint value of axis.

Syntax:

val_joint = JVal(#Joint_variable, joint_index)

Return value: val_joint

Data type: Number

Arguments: Joint_variable, joint_index

Data type : #Joint, index Number

Example:

Point #Joint_variable = Joint(10,30,50)

val_joint1 = JVal(#Joint_variable, 1)

val_joint2 = JVal(#Joint_variable, 2)

val_joint3 = JVal(#Joint_variable, 3)

Result:

val_join1 = 10

val_join2 = 30

val_join3 = 50

12.24. $Int16B – Converts Hex_code to 2-digit Ascii char

Description:

Converts Hex_code to 2-digit Ascii cha..

Syntax:

$hex_srt = $Int16B(hex_code)

Return value: $hex_str

Data type : String

Arguments: hec_code

Data type : Hex

Ex) hex_code = h^3132

Example:

hex_code = ^h3161

$hex_str = $Int16B(hex_code)

Result:

$hex_str: 1a

12.25. Len – Get length of string.

Description:

Get length of string

Syntax:

value1 = Len("CoreRobot")

Return value: value1

Data type : Number

Arguments: "CoreRobot"

Data type : $String

Example1:

length1 = Len("CoreRobot")

$corerobot = "thisiscorerobot"
length2 = Len($corerobot)

Result:

length1 : 9 length2 : 15

12.26. $Mid – Returns the specified string.

Description:

Returns a string containing n characters starting at the specified index.

Syntax:

$value1 = $Mid("CoreRobot", index1, count1)

Return value: $value1

Data type : $String

Arguments: "CoreRobot", index1, count1

Data type : $String, Number, Number

Example1:

count1 = 3

index1 = 2

$value1 = $Mid("CoreRobot", index1, count1)

Result:

$value1 : ore

12.27. Random - Return a random value.

Description:

Return a random value(0~1)

Syntax:

value1 = Random

Return value: value1

Data type : Number

Example1:

random1 = Random

Result:

Random1 = value of (0~1)

12.28. Round – Return rounded value.

Description:

Return rounded x **Syntax:**

value1 = Round(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number

Example1:

n1 = 3.6

round1 = Round(n1)

Result:

round1 : 4

12.29. Rx, Ry, Rz – Returns a rotating Trans matrix.

Description:

Returns a rotating Trans matrix, each argument rotated by x degrees..

Syntax:

Point tf = Rx(angle)

Point tf = Ry(angle)

Point tf = Rz(angle)

Return value:

Data type : Trans

Arguments: Rx, Ry, Rz

Data type : Number, unit: degree.

Example1:

angle = 90

Point tf = Rx(angle)

Result:

The transformation matrix defining Rx is $\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(angle) & -\sin(angle) \\ 0 & \sin(angle) & \cos(angle) \end{bmatrix}$,

So, Rx(90)=$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$

12.30. SDORead – Read the data through the SDO.

Description:

Read data to specific address.

Syntax:

Id : Slave Order (start order is 0)

Index : Address index(Enter the decimal value)

Sub-Index : Address sub-index(Enter the decimal value)

Length : byte length (1(8bit), 2(16bit), 4(32bit))

Value : Save the read data.

Ret-value : 0 normal state, -1 error state

Return value: value, Ret-value

Value - Data type : Number

Ret Value : -1, 0

-1 False, 0 True.

Example1:

Id = 0

Index = 12800 ;(0x3200)

Subindex = 0

Len = 2 ;(16 bit)

Value = 0

Retvalue = 0

SDORead id, index, subindex, len, value, retvalue

Result:

Value = 2 (read value)

Retvalue = 0

12.31. SDOWrite – Write the data through the SDO.

Description:

Write the data through the SDO.

Syntax:

Id : Slave Order (start order is 0)

Index : Address index(Enter the decimal value)

Sub-Index : Address sub-index(Enter the decimal value)

Length : byte length (1(8bit), 2(16bit), 4(32bit))

Value : input value.

Ret-value : 0 normal state, -1 error state

Return value: value, Ret-value

Value - Data type : Number

Ret Value : -1, 0

-1 False, 0 True.

Example1:

Id = 0

Index = 12800 ;(0x3200)

Subindex = 0

Len = 2 ;(16 bit)

Value = 3

Retvalue = 0

SDOWrite id, index, subindex, len, value, retvalue

Result:

Retvalue = 0 - 0x3200 00 16 input the value 3

12.32. Sig – Return AND operation of the Sig1, Sig2, ….

Description:

Return AND operation of the Sig1, Sig2, Sig3….

Syntax:

value1 = Sig(1001, 1002)

statussig = Sig(1007)

Return value: value1, statussig

Data type : Number

Ture(ON), False(OFF)

Arguments: 1001, 1002, 1007

Data type : Digital Input Number

Size of input: 1001~1128

Example1:

sig1 = Sig(1001, 1002)

statussig = Sig(1007)

sig1 returns, AND Operation of Sig(1001) and Sig(1002)

statussig returns Input sig(1007) status

Result:

sig1 : -1(TRUE) → 1001 = on, 1002 = on

sig1 : 0(FALSE) → 1001 = off, 1002 = on

12.33. Sin – Return Sin value.

Description:

Return Sin value, argument x unit is degree

Syntax:

value1 = Sin(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number, argument x unit is degree.

Example1:

sinedg1= Sin(90)

Result:

sinedg1 : 1

12.34. Sqrt – Return Square root value.

Description:

Return Square root value

Syntax:

value1 = Sqrt(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number

Example1:

n1 = 4

sqrt1 = Sqrt(n1)

Result:

sqrt1 = 2

12.35. SysTimer – Returns the SysTimer value corresponding to systimer id.

Description:

Returns the Timer value corresponding to timer id. Used with SetTimer, timer id can be from (1 to 9).). Set the time with SetTimer and measure the time elapsed using the Timer..

Syntax:

elapsed_time = SysTimer(timer_id, type)

Return value:

Data type : Number .

Arguments:

timer_id : Data type : Number 타이머 id

type: 1->usec, 2-> sec, 3->day

Example1:

```
SetSysTimer 1
WaitTime 0.1
usec = SysTimer(1, 1)
TPWrite 2, "elapsed = %f us", usec
SetSysTimer 2
WaitTime 1
sec = SysTimer(2, 2)
TPWrite 2, "elapsed = %f sec", sec
```

Use id 1(usec), if the elapsed time is greater than 1 second, skip seconds and return only micros, the same as sec, day timer id.

.

12.36. Timer – Returns the Timer value corresponding to timer id.

Description:

Returns the Timer value corresponding to timer id. Used with SetTimer, timer id can be from (1 to 9). Set the time with SetTimer and measure the time elapsed using the Timer.

Syntax:

elapsed_time = Timer(timer_id)

Return value:

Data type : Number .

Arguments:

timer_id : Data type : Number (timer id)

Example1:

```
SetTimer   1, 0
WaitTime 1
MoveJ #p1
SetDO 2
MoveL #p2
timer1 = Timer(1)
```

12.37. #TouchJoint/TouchTrans – Get TouchProbe detected postion

Description:

When touch (signal) is detected on IO for touch signal on motor driver, assign position value of current motor to Joint or Trans. Rather than using the WaitSig command and the Here command to get the position, you can get the correct position while the robot is running.

Syntax:

Point #p_value = #TouchJoint(Probe_index, Count_index)

Point t_value2 = TouchTrans(Probe_index, Count_index)

Return value: #p_value or t_value

Data type : #Joint or Trans

Arguments: Probe_index, Count_index,

Data type : Probe_index Number, Count index Number

Example1:

Point #Position1 = Joint(10,20,30,40,50,60)

TouchEnable TRUE, 1 → Enable 1 Rising then Falling type

TouchAxis 1 → 1 axis Monitoring

TouchStart 1, 1 → probe 1, 2 channel

MoveJ #position1 → detect signal while move to #position

TouchWait 1, result → if signal detected result = TRUE

TouchStop → Touch Scanning stop

IF result THEN

Point #touchr1 = #TouchJoint(1, 1) → probe1, Rising position, probe 1, count index 1

Point touchf1 = TouchTrans(1, 2) → probe1, Falling position, probe 1, count index 2

Point touchr2 = TouchTrans(2, 1) → probe2, Rising position, probe 2, count index 1

Point #touchf2 = #TouchJoint(2, 2) → probe2, Falling position, prove 2, count index 2

END

12.38. Trans – Create the Trans position.

Description:

create the Trans position variable by directly setting the value of the trans point.

Syntax:

Point value1 = Trans(x1, y1, z1, a1, b1, c1)

Return value: value1

Data type : Trans

Arguments: x1, y1, z1, a1, b1, c1

Data type : Number

Example1:

```
x1 = 10

y1 = x1

z1 = x1 + y1

a1 = x1^2

b1 = x1 * a4

c1 = 20

Point trans1 = Trans(x1,y1,z1,a1,b1,c1)
```

Result:

trans1 : 10, 10, 20, 100, 1000, 20

12.39. Translate – Find the Trans matrix that moves point p by dx dy, dz.

Description:

Find the Trans matrix that moves point p by dx dy, dz .

Syntax:

Point value1 = Translate(value2, dx1, dy1, dz1)

Return value: value1

Data type : Trans

Arguments: value2, dx1, dy1, dz1

Data type : Trans, Number, Number, Number

Example1:

Point nowposition = Trans(10,10,10)

dx1 = 10

dy1 = 20

dz1 = 10

Point newposition = translate(nowposition, dx1, dy1, dz1)

Result:

newposition : 20, 30, 20

12.40. Value – Convert String to number

Description:

Convert String to number.

Syntax:

value1 = value($value2)

Return value: value1

Data type : Number

Arguments: $value2

Data type : String

Example1:

$stringnum = "12233334444"

valuenum = Value($stringnum)

Result:

valuenum : 1223334444

13. Instruction Reference

Instructions for use of various commands of Macro Program

13.1. ABOVE/BELOW – Robot ABOVE and BELOW configuration

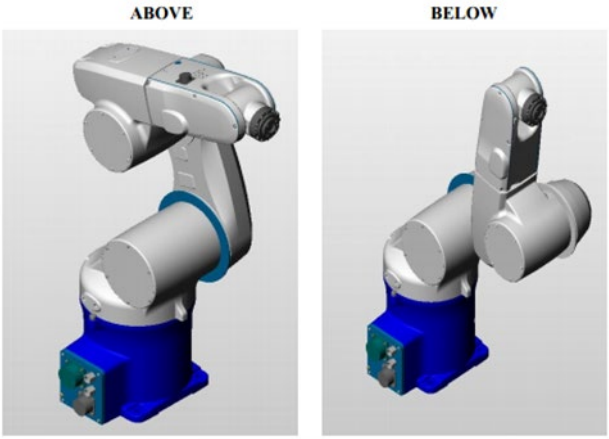
Description:

In singular position, its need to determine the direction of motion in 6-axis Robot. It decided by axis 2 and 3. ABOVE and BELOW configuration see the picture.

Syntax:

ABOVE

BELOW



13.2. Accel/Decel – Set acceleration and deceleration.

Description:

Default acceleration / deceleration may cause an unstable motion when the robot's weight moves with heavy objects, At this time, it is possible to change the acceleration / deceleration so that it can operate flexibly. Max speed is defined in ROBOT_CONFIGURE

If "Fixed" is not used, it applies only to the following motion commands.

Syntax:

Accel aspeed [Fixed]

Decel dspeed [Fixed]

Argumen ts: aspeed, dspeed, Fixed

Data type : Value Number, Value Number, Command

Value Number is the unit of Max Acceleration / Deceleration, The input range is **(**0.01 **~** 100%**)**

Example:

Accel 10

Decel 20

MoveJ #p1 ; Accel 10, Decel 20, when move to #p1 position

MoveJ #p2 ; Default Accel, decel when move to #p2

Accel 20 Fixed

Decel 20 Fixed

MoveJ #p3 ; Accel 20, Decel 20

MoveJ #p4 ; Accel 20, Decel 20

13.3. Accuracy – Sets the position reach recognition range.

Description:

When Robot moves to position1 -> 2 -> 3, if Accuracy value is set, robot does not pass position2, but moves smoothly to position3 by setting distance. The operation is determined by the combination of the deceleration and acceleration section, and the value must be set within the acceleration / deceleration section distance.

Unit is mm

Syntax:

Accuracy distance1 [Fixed]

Arguments: distance1, Fixed

Data type : value Number, Command

Example:

```
MoveL p1 ; default accuracy
Accuracy 20
MoveL p2 ; set accuracy 20
MoveL p3 ; default accuracy
MoveL p4 ; default accuracy
Accuracy 10 Fixed
MoveL p5 ; set accuracy 10
MoveL p6   ; set accuracy 10
MoveL p7   ; set accuracy 10
```

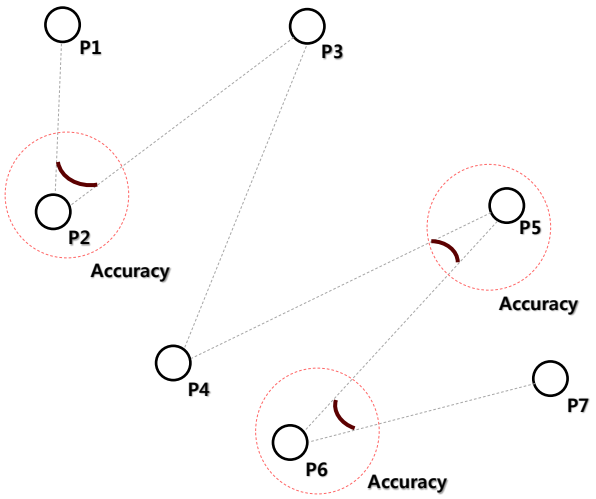


Figure 8 Accuracy motion

13.4. ALIGN – Align the z direction of the tool with nearest Base coordinate axes.

Description:

Align the z direction of the tool with nearest Base coordinate axes.

Syntax:

Align

Arguments: Align

move robot to align tool's z-axis with the base coordinate axes.

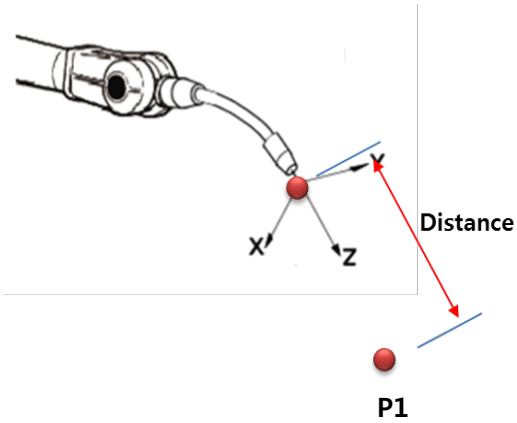
Example:

```
Point #joint1 = Joint(150,150,0)
MoveJ #joint1
Align ; Align the z-direction of the tool with the coordinate axis of the nearest base.
```

13.5. ApproJ – Joint move a distance with -Z direction of the tool coordinate system

Description:

Use the ApproJ command to move quickly to P1. Joint move a distance from p1 to the -Z direction of Tool.



Syntax:

ApproJ #joint1, distance1

ApproJ trans1, distance1

Arguments: #joint1 or trans1, distance1

Data type : #Joint or Trans, Number

The first argument is the reference coordinate, and the second argument is the distance from the reference point to the -Z axis direction of the Tool's coordinate.

Example:

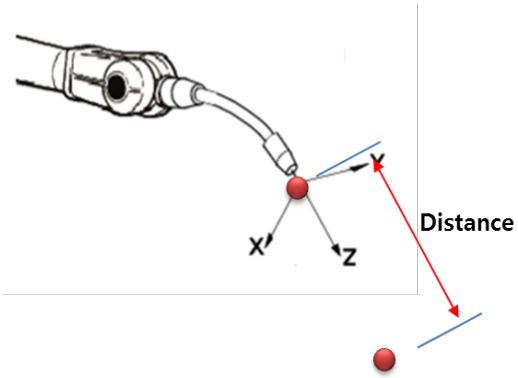
Point #joint1 = Joint(150,150,0)

ApproJ #joint1, 10 ; Move to 10 mm from #joint1 to -Z direction of the Tool coordinate.

13.6. ApproL – Linear move a distance with -Z direction of the tool coordinate system

Description:

Use the ApproJ command to move quickly to P1. Linear move a distance from p1 to the -Z direction of Tool.



Syntax:

ApproL #joint1, distance1

ApproL trans1, distance1

Arguments: #joint1 or trans1, distance1

Data type : #Joint or Trans, Number

The first argument is the reference coordinate, and the second argument is the distance from the reference point to the -Z axis direction of the Tool's coordinate

Example:

Point #joint1 = Joint(150,150,0)

ApproL #joint1, 10 ; Linear move to 10 mm from #joint1 to -Z direction of the Tool coordinate.

13.7. Brake – Stop Robot motion and proceed to the next step.

Description:

When Robot encounters a Brake during operation, it stops and executes the next step.

Syntax:

Brake

Arguments:

Data type : Command

Example:

```
MoveJ #p1
WaitTime 2
If Sig(1007) Then
  Brake
END
MoveJ #P2
```

If the digital input 1007 is on during operation, stop the current operation and execute the next step.

13.8. Break - Break the CP motion and move it to the correct position

Description:

Skip the accuracy, and move to the correct position.

Syntax:

Break

Arguments:

Data type : Command

Example:

MoveL p1 ;move to p1 with default accuracy

MoveL p2 ; move to p2 with default accuracy

MoveL p3 ; move to p3 with default accuracy

Break

MoveL p4 ; move to p4 without accuracy

MoveL p5 ; move to p5 with default accuracy

13.9. Call – Call program.

Description:

Call program

Syntax:

Call program

Arguments: prgoram

Data type : string

Example1:

Call examprg1 ; call program examprg1

Call examprg2 ; call program examprg2

.

13.10. Delay – Delay previous command for setting time.

Description:

Delay previous command for setting time. Delay = DelayTime + previous command.

Syntax:

Delay time1

Arguments: time1

Data type : Number (sec)

Example:

```
Point #joint1 = Joint(150,0,0)
Point #joint2 = Joint(150,150,0)
MoveJ #joint1
Delay 10
MoveJ #joint2
```

Move to #joint, send previous command(MoveJ #joint1) for 10 second.

13.11. DelayTime – Wait until the next command.

Description:

To move accurately to destination, wait for the next command for the setting value.

Syntax:

DelayTime time1

Arguments: time1

Data type : Number (sec)

Example:

```
Point #joint1 = Joint(150,0,0)
Point #joint2 = Joint(150,150,0)
Point #joint3 = Joint(0,150,0)
MoveJ #joint1 ; move to #joint1
DelayTime 0.1
MoveJ #joint2 ; move to #joint2, wait for 0.1 second until the next command(MoveJ #joint3)
MoveJ #joint3
```

13.12. DelayDO – Delay Digital Output for setting value

Description:

Set the output command Digital signal number and time to delay.

The time is counted during operation and DO output when the set time is over.

The output DO is held until reset.

Syntax:

DelayDO output1, time1

Arguments: output1, time1

Data type : Digital Output Number, value Number

DO can be set from 1 to 512.

Sets the delay time to time1. Unit is sec.

Example:

DelayDO 1, 0.5

DelayDO 2, 1

WaitTime 1

SetDO -1

SetDO -2

Program Start

DO signal 1 output when delay 0.5 second,
DO signal 2 output when delay 1 second.

Wait for 1 second

Both signals are getting off.

13.13. DepartJ – Robot moves -Z direction in the tool coordinate system

Description:

Robot moves according to the input distance in the -Z direction of the tool coordinate system.

Syntax:

DepartJ distance1

DepartJ distance1

Arguments: distance1

Data type : Number

Distance to move –Z direction

Example:

```
Point #joint1 = Joint(150,150,0)

MoveJ #joint1

DepartJ 10
```

Moves from #joint1 position to Z direction by -10 based on tool coordinate.

13.14. DepartL – Robot moves linearly -Z direction in the tool coordinate system

Description:

Robot moves linearly according to the input distance in the -Z direction of the tool coordinate system.

Syntax:

DepartL distance1

DepartL distance1

Arguments: distance1

Data type : Number

Distance to move –Z direction

Example:

```
Point #joint1 = Joint(150,150,0)

MoveJ #joint1

DepartL 10
```

Moves linearly from #joint1 position to Z direction by -10 based on tool coordinate.

13.15. Drive – Moves the selected axis

Description:

Moves the specified axis at the current position by the entered value.

Syntax:

Drive delta_joint, value1

Arguments: delta_joint, value1

Data type : Delta Joint Number, value Number

Joint Number : Axis you want to move

value Number : Moving distance.

Example:

Point #joint1 = Joint(20,20,20)

MoveJ #joint1

Drive 1, 20

Drive 2, 10

Drive 3, 10

Move to MoveJ #joint1 position

Move 1st joint 10 degree. Desired position is (30, 20, 20,)

Move 2nd joint 10 degree. Desired position is (30, 30, 20,)

Move 3rd joint 10 degree. Desired position is (30, 30, 30,)

.

13.16. FMoveL – Fixed FTool works with uniform distance.

Description:

Works with uniform distance to specified FTool (Fixed Tool).

Syntax:

FMoveL #jvar1

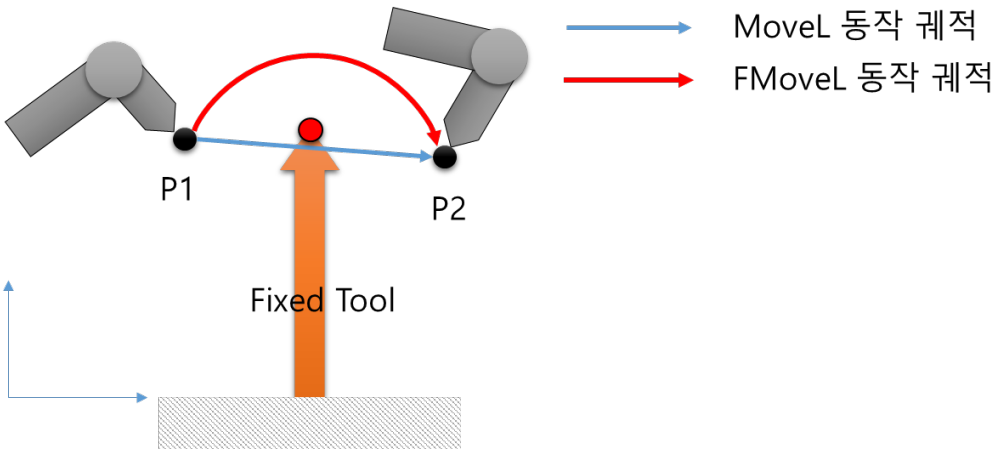
FMoveL tvar1

Arguments: #jvar1 or tvar1

Data type : #Joint or Trans

Example:

```
FTool Trans(1100,0,750,0,173,0)
Point #pf1 = Joint(0,-6,53,0,16.7,0)
Point #pf2 = Joint(0,29.4,12.2,0,94,0)
MoveJ   #pf1
WaitTime 1
TPWrite 2,"FMoveL"
FMoveL   #pf2
FMoveL   #pf1
WaitTime 1
TPWrite 2,"MoveL"
MoveL   #pf2
MoveL   #pf1
```



13.17. FTool – Set the FTool(Fixed Tool).

Description:

FTool This command sets the coordinate system.

Syntax:

FTool Trans(x1, y1, z1, a1, b1, c1)

Arguments: x1, y1, z1, a1, b1, c1

Data type : Number

Enter the Trans coordinate value.

When 0, 0, 0 is input, the euler angle value representing the attitude is the identity.

Example:

FTool Trans(1100,0,750,0,173,0)

13.18. HALT – Stop the Program.

Description:

Stop the program at the desired step.

Syntax:

HALT

Example:

```
Point #p1 = Joint(10,20,30)
Point #p2 = Joint(40,50,60)
MoveJ   #p1
HALT
MoveJ   #p2
```

Moves to MoveJ #p1.

Stop program.

The program is aborted and the movej # p2 command is not executed.

13.19. HOME/HOME2 – Moves specified Home position.

Description:

Moves to the specified Home location.

you can specify the Home and Home2 in setting home page.

Syntax:

Home

Home 2

Arguments: HOME or HOME 2

Moves to the location specified by the Home position. To specify the Home position, you can use the Setting -> Home page.

Example:

```
HOME

Point #joint1 = Joint(150,150,0)

MoveJ #joint1

HOME 2
```

Use HOME to move to the originally assigned to HOME

Move the robot with MoveJ, move to HOME2 position, and finish the operation.

13.20. IncJ – Moves by set Joint.

Description:

The specified axis moves by the specified number from the current position. If you enter 0, the corresponding joint will not move.

Syntax:

IncJ delta_joint1, delta_joint2, ...

Arguments: delta_joint1, dalta_joint2

Data type : Delta Joint Number

The values of each delta joint contained.

Example:

delta_joint1 = 10

delta_joint2 = 10

delta_joint3 = 10

MoveJ #p1

IncJ delta_joint1, 0, 0

IncJ 0, delta_joint2, 0

IncJ 0, 0, delta_join3

Move position #p1.

Move 1st joint 10 degree.

Move 2nd joint 10 degree.

Move 3rd joint 10 degree.

13.21. IncL – Moves linear interpolation for the base coordinate system.

Description:

Moves linearly in the base coordinate system by the number entered.

Syntax:

IncL delta_x, delta_y, delta_z

Arguments: delta_x, delta_y, delta_z

Data type : Delta Trans matrix Number

Example:

```
delta_x = 10
delta_y = 20
delta_z = 30
MoveJ #p1
IncL delta_x, 0, 0
IncL 0, delta_y, 0
IncL 0, 0, delta_z
```

Move to the location corresponding to #p1. Then, through the IncJ instruction, it moves delta x, 10, and then delta y 20, and finally delta z 30, at the current position. When moving, it moves to linear interpolation.

13.22. IncT – Moves based on Tool coordinate.

Description:

Moves from the current position by the value entered in the tool coordinate system.

Syntax:

IncT delta_tx, delta_ty, delta_tz, ...

Arguments: delta_tx, delta_ty, delta_tz, ...

Data type : Delta Trans matrix Number

Tool coordinate base Trans matrix x, y, z

Example:

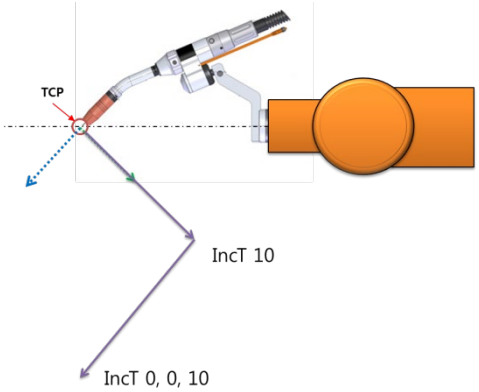
MoveJ #p1

IncT delta_tx, 0, 0

IncT 0, delta_ty, 0

IncT 0, 0, delta_tz

Move to position #p1, Based on tool coordinates, delta tx moves 10The next delta ty moves 10, the delta tz moves 10 at the end.



13.23. LEFTY/RIGHTY – Select the robot geometry

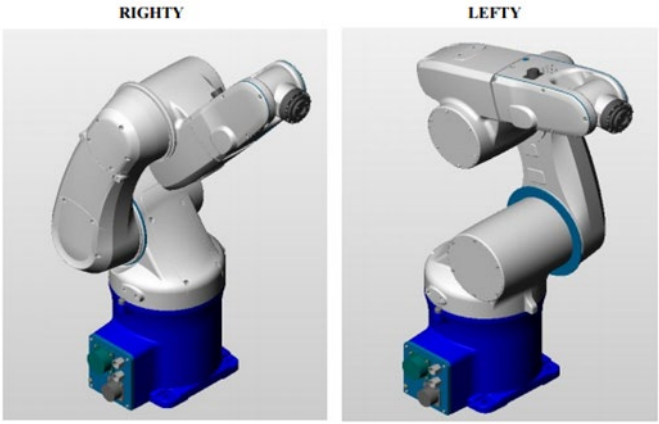
Description:

You need to determine the direction of motion when you are in a certain boundary position in the 6-axis Robot. When the direction is not determined, it keeps the previous direction. If the shape of the right arm is RIGHTY and the shape of the left arm is the shape of the robot, LEFYT is used to determine the shape. If the robot is symmetrical, there is no point in determining the shape.

Syntax:

LEFTY

RIGHTY



13.24. MakeStr – Make String variable.

Description:

Makes various types of variables as String.

Syntax:

MakeStr $strval, dataformat, data

Arguments: $strval, dataformat, data

$strval: String name

dataformat: s, f, d …

data: value

Example:

Point #jcur = #Here

Decompose jval[0] = #jcur

MakeStr $msgsend, "Position : %.3f, %.3f, %.3f, %.3f", jval[0], jval[1], jval[2], jval[3]
Get current position using #Here command.
Get each joint value using Decompose command.
Make $msgsend variable. (Position : 12.345, 12.345, 12.345, 12.345)

13.25. MoveC – Moves the robot circularly.

Description:

MoveC is used to move the tool center point (TCP) circularly to a given destination.

Syntax:

Movec passjoint, endjoint

Movec passtrans, endtrans

Arguments: passjoint or passtrans, endjoint or endtrans

Data type : #Joint or Trans, #Joint or Trans

Example:

```
MoveJ #joint1

MoveL trans1

MoveC passtrans, endtrans
```

The robot is in the position trans1.c

Starting from the position A, passing through the position B, and the operation is completed at the position C.

13.26. MoveH – Pass the Singular position in linear movement (Only Serial6 robot).

Description:

When the robot passes through the Singular section, the speed of the motor suddenly increases and an Over speed limit error occurs. This is a command to avoid the phenomenon and reduce the speed of the robot to pass through the singular section. However, because of the speed control, the singular section is avoided, which may not be perfect in some cases.

To use the MoveH command, you need to add the following option to corecon.conf.

Avoid Teach mode singular : TEACH_SINGULAR = TRUE

TEACH_SINGULAR_SPEED_LIMIT = 0.001~1.0 (if value is 1.0 , Apply the max manual speed limit when move in singular section)

Avoid Repeat mode singular : REPEAT_SINGULAR = TRUE

REPEAT_SINGULAR_SPEED_LIMIT = 0.001~1.0(if value is 1.0, Apply the max speed when move in singular section.)

Therefore, it is necessary to adjust the setting values of AXIS Specification items AXIS_MAXRPM, AXS_RPM etc.

Syntax:

MoveH tvar1

Arguments: #jvar1 or tvar1

Data type : Trans

Example:

```
Point tvar1 = Trans(1270,0,1550,0,90,0,0)
Point tvar2 = Trans(1270,0,1570,0,90,0,0) ;singular point
Point tvar3 = Trans(1270,0,1590,0,90,0,0)
MoveJ   tvar1
MoveH   tvar2
MoveH   tvar3
```

13.27. MoveJ – Moves the robot by Joint movement.

Description:

MoveJ is used to move the robot quickly from one point to another when that movement does not have to be in a straight line

Syntax:

MoveJ #jvar1

MoveJ tvar1

Arguments: #jvar1 or tvar1

Data type : #Joint or Trans

Example:

```
Home

Point #j1 = Joint(10,10,10)

Point t1 = Trans(10,10,10)

MoveJ #j1

MobeJ t1
```

13.28. MoveL – Moves the robot linearly.

Description:

MoveL is used to move the tool center point (TCP) linearly to a given position variable.

Syntax:

MoveL #jvar1

MoveL tvar1

Arguments: #jvar1 or tvar1

Data type : #Joint or Trans

Example:

```
Home

Point #j1 = Joint(10,10,10)

Point t1 = Trans(10,10,10)

MoveL #j1

MoveL t1
```

13.29. MoveX – Monitor the specified signal during the Movel motion

Description:

If you receive a signal from the specified DI during the linear movement, stop the current movement and execute the next step from the position where the signal was received.

Syntax:

MoveX trans1, 1007

Arguments: #joint1, 1007

Data type : trans1, Input Signal Number

The first argument is the position variable input, the second argument is the DI number.

Example:

```
Point trans1 = trans(10,10,0)

Point trans2 = trans(150,150,0)

MoveX trans1, 1007

MoveL trans2
```

Moving to trans1 according to MoveX command, When DI 1007 receives an input signal, Stops the MoveX command and execute the next command MoveL move to trans2 position..

13.30. MoveXJ – Monitor the specified signal during the Movej motion

Description:

If you receive a signal from the specified DI during the Joint movement, stop the current movement and execute the next step from the position where the signal was received.

Syntax:

MoveXJ #joint1, 1007

Arguments: #joint1, 1007

Data type : #Joint, Input Signal Number

The first argument is the joint variable input, the second argument is the DI number.

Example:

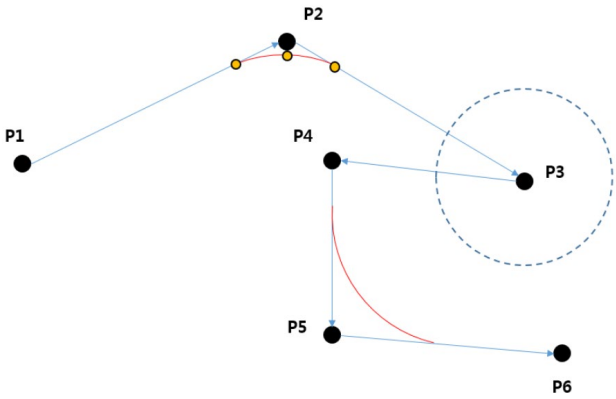
```
Point #joint1 = Joint(200,200,0)

Point #joint2 = Joint(150,150,0)

MoveXJ #joint1, 1007

MoveJ #joint2
```

Moving to # joint1 according to MoveXJ command**,** When DI 1007 receives an input signal, Stops the MoveXJ command and execute the next command MoveJ move to #joint2 position**.**

13.31. MovePi, MoveP – Move multiple points in constant velocity.

Description:

This command is used when moving several points at a constant speed. It moves to the start point and the end point of constant velocity by one line.



Move the section from P1 to P6 at constant speed as shown.

Syntax:

MovePi joint1[trans coordinate], R[radius] --- Start & Passing point

MoveP joint2[trans coordinate], R[radius] --- Start & End point

Arguments: Joint1, Joint2, R

Data type : Joint1, Joint2 = Trans Point

R = Number

Example:

```
MoveJ P1
MovePi P2, 10    ;   radius = 10
MovePi p3, 0     ; radius = 0
MovePi p4, 0
MovePi p5, 20
MoveP p6, 0      ; The last point does not apply a radius, so set to 0.
```

13.32. MSig – IO signal is output during Motion step.

Description:

If you set MSig before the motion command, Output signal execute during the moving state.

Syntax:

MSig signal_num MSTIME time_sec

MSig signal_num MSDIST dist_mm

MSig signal_num MSPERC percent

Arguments: signal, option, value

Data type : Number, MSTIME or MSDIST or MSPERC, value

Number is the Name of Dout. + Is signal On, and - is signal Off..

option : MSTIME(Time unit is sec), MSDIST(Distance unit is mm), MSPERC(Percentage %).

value : Set time, distance, percentage (%) value according to option.

Example:

MSig 20, MSTIME, 1

MSig 21, MSTIME, 2

MoveJ Joint(10,20,30)

MSig 22 MSDIST, 100

MSig -20, MSDIST, -100

MoveJ Joint(0,0,0)

MSig 23, MSPERC, 30

MSig -21 MSPERC -30

MoveJ Joint(10,20,30)

Reset

Dout 20 output after 1 second of "MoveJ Joint(10,20,30)" operation.

Dout 21 output after 2 second of "MoveJ Joint(10,20,30)" operation.

Dout 22 output after moving 100mm of "MoveJ Joint(0,0,0)" command.

Dout 20 tune off when robot reach 100mm to the "MoveJ Joint(0,0,0)" destination.

Dout 23 output after moving 30% of "MoveJ Joint(10,20,30)" command

Dout 21 tune off when robot reach 30% to the "MoveJ Joint(10,20,30)" destination.

Reset all the signal.

●Please note that MSig commands are checked in order when using MSig command.

Ex:

```
MSig   -15, MSDIST, -0.1
MSig   15, MSDIST, 0.1
MoveJ #joint1
```

If you program in the above order, DO 15 will not turn OFF at the last end point.

13.33. OverDI – Control the digital input signal.

Description:

You can turn on/off the digital input during Macro program operation.

Syntax:

OverDI On/Off

Example:

OverDI ON

SetDI 1002 ; DIn 2 On

WaitTime 1

SetDI -1002 ; DIn 2 Off

OverDI OFF

13.34. PrefetchSig – Contral the PREFETCH_SIGNAL command.

Description:

Sets the output timing of the signal.

When PrefetchSig On, DI and DO are output when the previous robot motion starts.

When PrefetchSig Off, DI and DO are output when the previous robot motion complete.

.

Syntax:

PrefetchSig On/Off

Example:

```
PrefetchSig ON
MoveJ #p1
SetDO 10
PrefetchSig OFF
MoveJ #p2
SetDO -10
```

Signal output(SetDO 10) when start MoveJ #p1 motion.

Signal output(SetDO 10) when complete MoveJ #p2 motion.

13.35. Reset – Turn off all the output signal.

Description:

This command turns off all digital and analog signals.

Syntax:

Reset

Arguments:

Data type : Command

Example:

SetDO 1

SetDO 2

WaitTime 1

Reset

Turn on digital output signal number 1 and 2.

Hold the output signal for 1 second.

Turn off digital output signal number 1 and 2.

13.36. RunMask – Option to export signal only at running process.

Description:

The IO signal continues to output the IO currently being output when the program is stopped.

You can turn off the output using the Run mask function.

Set the signal to RunMask, when you want to turn off the output when the program stops.

The DO signal is Off when Macro is on time. You can check the operation status of Macro separately.

Syntax:

RunMask startdo, count1

Arguments: startdo, count1

Data type : Digital Output Number, value Number

DO can be set from 1 to 128.

The value is a continuous count for the output signal.

Example:

RunMask 6, 4

MoveJ #p1

MoveJ #p2

As the RunMask is declared, Four DO from DO6 to 7, 8, and 9 are declared

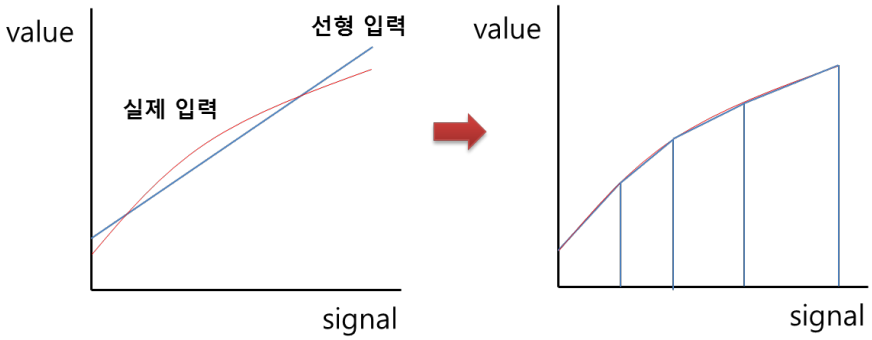
When the macro is running, DO6 ~ 9 are output. When Hold or Macro finishes Output is off.

13.37. SetAICalib – Set the calibration table about Analog input channel

Description:

Sets the relationship between the input signal level of the Analog and the actual physical value.

Analog physical values often do not change linearly as shown below. In this case, you can use the calibration table to divide into several linear segments to improve the accuracy of the output. Must be enter the minimum and maximum values.



Syntax:

SetAICalib channel#, signal1, value1, signal2, value2, … , signal_N, value_N

Arguments: channel#, signal, value

Channel# : 1 to 128 channels can be set.

Actual output is limited to the number of device channels installed.

Signal, value : Enter the pair of signal and value that make up the calibration table.

Up to 20 entries can be entered.

Example:

SetAITable 1, 0, 100, 4, 200, 6, 300, 10, 400

Define the calibration table.

index	signal	value
1	0	100
2	4	200
3	6	300

4 10 400

Temp = AIN(1)

If analog input voltage of channel 1 becomes 6v, temp becomes 300.

13.38. SetAO – Output the Analog signal.

Description:

Set and output analog output channel

The output value is set to the value defined in the calibration table. Refers the SetAOCalib command for calibration table definitions

Syntax:

SetAO channel#, value

Arguments: channel#, value

Channel# : 1 to 128 channels can be set.

Actual output is limited to the number of device channels installed

Value : Output analog value

Example:

SetAO 1, 30

WaitTime 1

SetDO 1, 0

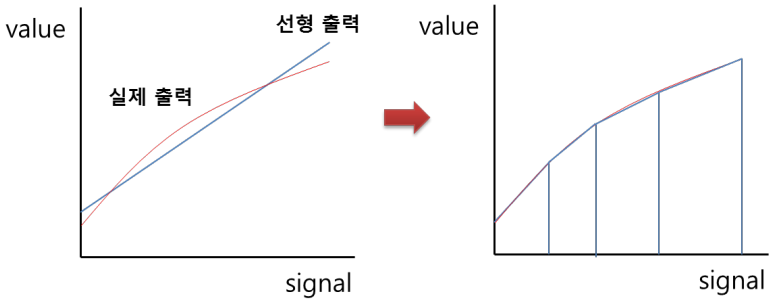
Set analog value of channel 1 to 30 and then set it to 0. The values(30) are converted to signal values by referring to the calibration table.

13.39. SetAOCalib – Set the calibration table about Analog output channel

Description:

Sets the relationship between the output signal level of the Analog and the actual physical value.

Analog physical values often do not change linearly as shown below. In this case, you can use the calibration table to divide into several linear segments to improve the accuracy of the output. Must be enter the minimum and maximum values.



Syntax:

SetAOCalib channel#, signal1, value1, signal2, value2, … , signal_N, value_N

Arguments: channel#, signal, value

Channel# : 1 to 128 channels can be set.

Actual output is limited to the number of device channels installed

Signal, value : Enter the pair of signal and value that make up the calibration table.

Up to 20 entries can be entered.

Example:

SetAOTable 1, 0, 100, 4, 200, 6, 300, 10, 400

Define the calibration table.

index	signal	value
1	0	100
2	4	200
3	6	300
4	10	400

If analog output value 300, analog output voltage value is 6v.

13.40. SetDO – Digital Output.

Description:

Set the digital output number to be output

Position number set output, Negative number reset the output.

Syntax:

SetDO output1

Arguments: output1

Data type : Digital Output Number

In the system can be use 1~512 digital output.

Example:

SetDO 1

SetDO 2

WaitTime 1

SetDO -1

Digital output signal 1, 2 output

Wait for 1second.

Reset the signal 1output.

13.41. SetJ – Set the Joint variable data.

Description:

Set the Joint variable data.

Syntax:

SetJ #jval, j1, j2, j3, j4, j5, j6

Arguments: #jval, j1 ~ 6

#jval : joint variable

j1~6 is same as Joint1~ 6 order.

Example:

```
Point #jval = Joint(0,0,0,0,0,0)
SetJ #jval, 10, 10, 10, 10, 10, 10
MoveJ   #jval
```

Define Joint variable #jval.

Set the #jval j1~j6 to 10.

Move to #jval(Joint(10,10,10,10,10,10)) position.

13.42. SetP – Set the variable Trans value.

Description:

Set the variable Trans value.

Syntax:

SetP tval, x, y, z, a, b, c

Arguments: tval, x, y, z, a, b, c

tval : trans variable

Enter the Trans value. When 0, 0 or 0 is input, The euler angle value representing is the identity. Untyped values are set to zero.

Option: /X, /Y, /Z, /X/Y/Z, /X/Y, /X/Z, /Y/Z, /ABC

SetP/X: Change the transX coordinate of the variable.

SetP/Y: Change the transY coordinate of the variable.

SetP/Z: Change the transZ coordinate of the variable.

SetP/X/Y/Z: Change the transRXYZ coordinate of the variable (/X/Y), (/X/Z), (/Y/Z) also possible

SetP/ABC: Change the transABC coordinate of the variable.

Example1:

```
Point tval = trans(1270,0,1570,0,90,0)
SetP tval, 1280, 0, 1570, 0, 90, 0
MoveJ   tval
```

Define the trans value tval.

Change the trans value tval as 1280, 0, 1570, 0, 90, 0.

Move to tval(trans(1280,0,1570,0,90,0)).

Example2:

```
Point tval = trans(1270,0,1570,0,90,0)
SetP/X/Y/Z tval, 1000, 200, 1500
MoveL tval
```

Change the Trans value tval x, y, z position only (The values of a, b, and c are maintained).

13.43. SClose – Close the serial communication.

Description:

Cliose serial the serial communication.

Syntax:

SClose sid

Arguments: sid

sid: serial communication id

Example:

sid = 0

SClose sid

13.44. SOpen – Open the serial communication.

Description:

Open the serial communication.

Syntax:

SOpen sid, port_num, baudrate , databit , stopbit , paritybit , ret_flag

Arguments: sid, port_num, baudrate, databit, stopbit, paritybit, ret_flag

sid: serial communication id

port_num: 1-debug port, 2-rs232 port, 3-rs485 port, 4, rs232 port

baudrate: number

stopbit: number

paritybit: number

ret_flag: 0/sucsses, /false

Example:

sid = 0

port_num = 3

baudrate = 9600

databit = 8

stopbit = 1

paritybit = 0

ret_flag = 0

SOpen sid, port_num, baudrate , databit , stopbit , paritybit , ret_flag

13.45. SRead – Read Serial data.

Description:

Read Serial data.

Syntax:

SRead sid, $str_read, ret_flag

Arguments: sid, $str_read, ret_flag

sid: serial communication id

$str_read:

ret_flag: 0/sucsses, /false

Example:

sid = 0

port_num = 3

baudrate = 9600

databit = 8

stopbit = 1

paritybit = 0

ret_flag = 0

SOpen sid, port_num, baudrate , databit , stopbit , paritybit , ret_flag

SRead sid, $str_read, ret_flag

13.46. SRead2 – Read serial data of selected section.

Description:

Read serial data of selected section.

Syntax:

SRead2 sid,$strvalue, $startcode, $endcode, [ret_flag], [timeout]

Arguments: sid, $strvalue, $startcode, $endcode, ret_flag, timeout

sid: serial communication id

$strvalue:

$startcode:

$endcode

ret_flag: 0/sucsses, /false

timeout:

Example:

```
sid = 0
port_num = 3
baudrate = 9600
databit = 8
stopbit = 1
paritybit = 0
ret_flag = 0
SOpen sid, port_num, baudrate , databit , stopbit , paritybit , ret_flag
$strvalue = ""
$startcode = "a"
$endcode = "c"
timeout = 2
SRead2 sid,$strvalue, $startcode, $endcode, ret_flag, timeout
```

13.47. SWrite – Write serial data.

Description:

Write serial data

Syntax:

SWrite sid, $msgwrite, ret_flag

Arguments: sid, $msgwrite, ret_flag

sid: serial communication id

$msgwrite:

ret_flag: 0/sucsses, /false

Example:

```
sid = 0
port_num = 3
baudrate = 9600
databit = 8
stopbit = 1
paritybit = 0
ret_flag = 0
SOpen sid, port_num, baudrate , databit , stopbit , paritybit , ret_flag
$strvalue = "test"
SWrite sid, $strvalue, ret_flag
```

13.48. SINGLE/DOUBLE – Determine the rotation range of the 6 axis.

Description:

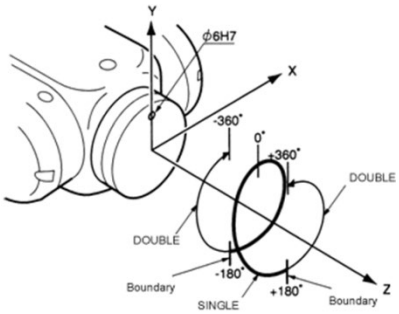
If you are using a tool or geometry that can rotate 6 axis one turn, select SINGLE

if you are using a tool or geometry that can rotate 2 wheels, select DOUBLE.

Syntax:

SINGLE

DOUBLE



13.49. Smooth – S-curve motion.

Description:

The acceleration / deceleration section of the robot is implemented with S-curve to smooth motion..

Syntax:

Smooth TRUE/FALSE, [jerk time]

Arguments: speed1, Fixed

Data type : **Bool,** Value Number

Bool : Enter TRUE to turn S-curve motion on, FALSE to off.

Value : Enter the jerk value of S-curve motion to set S-curve motion.

Example:

```
Smooth TRUE, 0.3
MoveL pt100        ; S-curve motion is applied when moving to pt100 position.
Smooth FALSE, 0
MoveL pt200        ; Trapezoidal motion is applied when moving to pt200 position.
```

13.50. Stable – The robot stick the specified position for the set time.

Description:

Delay and Stable are motion commands.

The Delay and Stable commands created after the Move command wait until the movement ends. Delay only performs the task waiting after the previous move, but Stable continues to execute the command to move to the target position for the given time (MoveJ current position + stable time). This stabilizes robot motion and increases accuracy. Non-motion commands are executed after Delay and Stable commands.

Syntax:

Stable time1

Arguments: time1

Data type : Value Number

Enter the desired delay time. Unit is sec

Example:

```
MoveJ #joint1
Stable 1
MoveJ #joint2
```

After the robot moves to # joint1, it stick command for one second. If the robot is out of the specified error range in the draft, the Wait time is changed back to 1 second and does not move to the # joint2 position. Be careful when using it because it moves infinitely until it reaches the correct position.

13.51. StableTime – Use the Stable function for motion commands.

Description:

Unlike the motion instruction Stable, StableTime is a non-motion instruction.

It is applied with motion commands just above stabletime.

It is a command to use Stable function in motion command.

Syntax:

StableTime time1

Arguments: time1

Data type : Value Number

Enter the desired delay time. Unit is sec

Example:

MoveJ #joint1

StableTime 1

MoveJ #joint2

MoveJ #joint3

The robot moves to # joint1.

The robot moves to #joint2. Sends a command to keep # joint2 position for 1 second. If the robot is out of the specified range in one second, the Wait time changes back to 1 second and does not move to next command.

Be careful when using it because it moves infinitely until it is held at the selected position for 1 second.

13.52. SPEED – Set the operation speed.

Description:

Set the robot's motion speed.

When Fixed is set, the speed apply the all motion commad. If Fixed is not used, the speed **apply only** next motion command.

Syntax:

Speed speed1 [Fixed]

Arguments: speed1, Fixed

Data type : Value Number

If Fixed is not used, the speed apply only next motion command.

There are three types of speed1 units, mm/sec, mm/min and%.

The default unit is%.

If you want to use mm / sec, mm/min, enter the unit.

Example:

Speed 10

MoveJ #joint1

MoveJ #joint2

Speed 20mm/s Fixed

MoveJ #joint3

MoveJ #joint4

Set to 10% of full speed in first step, move to 10% speed with # joint1

Move to default speed with #joint2,

All motion command speed set by 20mm/sec,

Move to 20mm/sec with # joint3, move to 20mm/sec with # joint4.

(Caution) When using Speed command as%, the operating speed does not change in units of speed due to acceleration / deceleration time. For example, if the time required from p1 to p2 is 2sec with

Speed 100, the time required from p1 to p2 is 4sec and the speed does not become half of speed 100%. To change the operating speed in multiples of Speed, you can set SPEED_RATE_TYPE = 1 in the robot config file (the default value is 0).

13.53. SubAbort – Stop the Subtask.

Description:

Stop the Subtask.

Syntax:

SubAbort subtask

Arguments: SubAbort, subtask

Data type : Value Number

Subtask number to stop (1~4)

Example:

SubAbort 1

Stop the Subtask 1.

13.54. SubExecute – Execute Subtask.

Description:

Execute Subtask.

Syntax:

SubExecute task number, program name, cycle, step

Arguments: SubExecute, task number, program name, cycle, step

Data type : Number, string, Number, Number, Number

Subtask number(1~4)

Name of Sub program

Repeat Cycle order

Start line number

Example:

SubExecute 1, pgsub1, 10, 2

In subtask 1, execute pgsub1 program 10 times from the second line

13.55. UWRIST/DWRIST – Determines the shape of WRIST in ROBOT

Description:

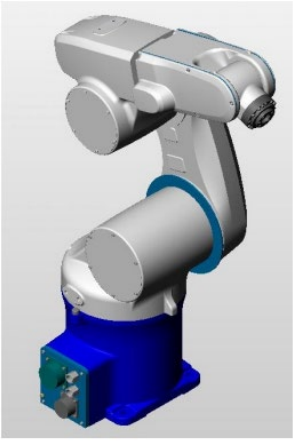
If the wrist shape of the robot is bent upward, UWRIST.

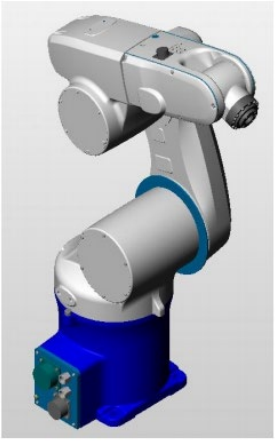
If the wrist is bent downward, select DWRIST.

It is determined by Joints 4 and 5.

Syntax:

UWRIST/DWRIST





13.56. Wait – Waits until the condition is satisfied or the set time is over.

Description:

Wait until the condition is satisfied or the set time is over.

In the waiting state, when the condition is satisfied, it is released and Result Flag result value is True.

However, when waiting is released by TimeOut, Result Flag becomes False.

You can distinguish between Signal and timeOut through Result Flag.

Syntax:

WAIT condition1, time1, result

Arguments: condition1, time1, result

Data type : Digital Input Number or function sig, Value Number, Result Number

DI is available from 1001 to 1512, you can use the sig function.

Value Number unit is seconds.

Result is 0 and -1, that is used in the if statement.

Example:

```
Wait 1001, 3.5, result1
If result1 then
    MoveJ #p1
    Wait Sig(1001, 1002) 3.5, result2
      If result2 then
          MoveJ #p2
      End
End
```

Wait 3.5 seconds for the DI 1001 signal.

If signal is on, result1 is true. If statement checks result1 state and executes MoveJ # p1 when the result value is True. Or not, Wait 3.5 seconds.

When DI 1001 and 1002 signals are all On, result2 is true, if statement is true and run MoveJ # p2.

13.57. WaitTime – Waits for the entered time.

Description:

Waits for the entered time without executing the next command

Syntax:

WaitTime timevalue

Arguments: timevalue

Data type : Time Number

wait while timevalue.

13.58. WaitSig – Wait until the condition is satisfied

Description:

Wait until signal is input to digital input,

When signal input is received, it is released from standby state and it moves to the next step.

Syntax:

Wait Sig(DI, DI), waittime, result

Arguments:

Data type : Digital Input Number, Bool Type

DI can be used from 1001 to 1128.

Waittime is the maximum time to wait.

result waits for the DI signal during the waittime and the result value is input.

true if signal is input; false otherwise

Example:

```
Wait Sig(1001, 1002), 2, result
IF result then
MoveJ #p1
end
```

Do not move to the next step until the signal is On(DI 1001, 1002).

If a signal is input to DI 1001 or 1002, the value of result is true.

Execute MoveJ # p1 when the if condition is satisfied.

CL Programming Manual

Copyright © 2016–2017 CoreRobot

All rights reserved.

Printed in the Republic Of Korea

#603, IT MIRAE Tower, 33, Digital-ro 9-gil,

Geumcheon-gu, Seoul, Republic Of Korea

CoreRobot Inc.

Web: www.core-robot.com

Tel: 82-2-2027-6565

Fax: 82-2-2027-6565