CL Programing Manual

CORE ROBOT

2016. 7. 29.

Contents

1. Introduction 5

2. Data & Variables 6

   1) Numeric 6

   2) String 7

   3) Joint Location 7

   4) Trans Location (Cartesian Coordinate) 8

   5) Array 10

   6) Unit 10

3. Constant 11

4. Operator 12

   1) Unary Real Operators 12

   7) Binary Real Operators 12

5. Comment : ; 13

6. Controlling the program flow 14

   1) Goto 15

   2) CALL 15

   3) Signal Interrupt 16

   4) Return, Stop, Pause 16

   5) Wait 17

   6) IF .. THEN .. ELSE .. END 18

   7) Looping structures : While ... Do, Do ... Until, For 19

   8) SWITCH ... CASE DEFAULT END 20

7. Functions 22

8. Instructions 25

1) Transform & Position 25

2) Robot Motion 27

3) Motion Modifiers 30

4) I/O 33

5) Touch Probe 34

6) Network 35

7) Conveyor Tracking 36

8) ETC 42

9) User coordinate instructions 42

9. Diagnostic Instructions 43

10. Program selection with an external i/o 44

1) Example : Program selection with EPS MODE 45

2) Program selection with Bits instruction and SWITCH CASE structure 45

11. Example programs 47

1) Basic motion program 47

2) CP Motion 48

3) IO Instructions 48

4) MoveC 49

5) Conveyor Tracking and Pickup 50

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.

1) Numeric

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:

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

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.

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
lampA = ON
foundOK = TRUE
```

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"
```

3) Joint Location

A joint location's 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 locations: Playback 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.

```
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
```

Like 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.

4) Trans Location (Cartesian Coordinate)

A transformation location 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 locations: 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 locations: 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
```

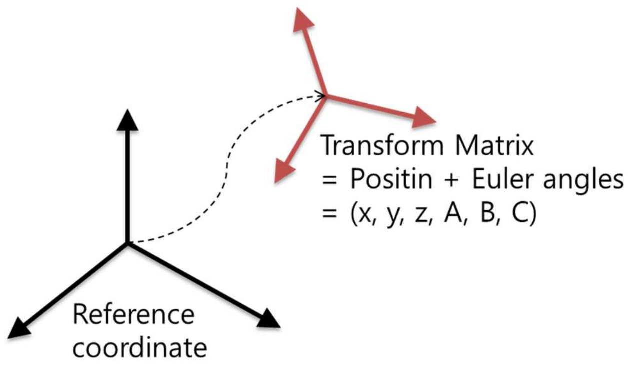


Figure 1 Trans location data

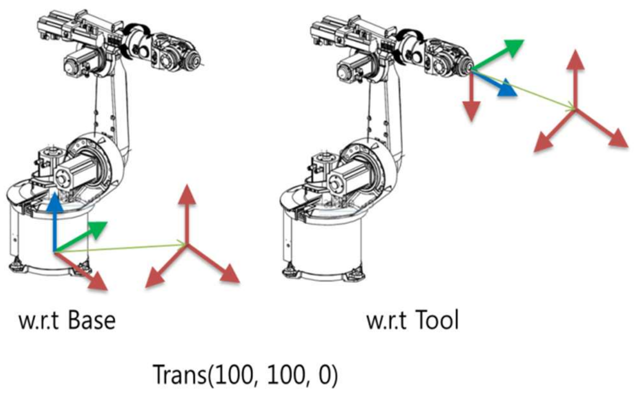


Figure 2. Actual location depends on the reference coordinate

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.

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
```

6) 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 : 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 constant
  - 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

1) Unary Real Operators

- **COM** : Complement
- **-** : Negation
- **NOT**

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

B = COM Aa ; Aa becomes 0xFFFFFFFC

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

7) Binary Real Operators

- **^** : power,
- ***** , **/** , **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. 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
```

6. 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

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            ; Label 100으로 분기
        END
```

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

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

2) CALL

- **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.

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

3) Signal Interrupt

- **Interrupt** Signal#, InterruptHandler, [Motion Stop = TRUE]

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
```

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.
```

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"
```

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
```

6) IF .. THEN .. ELSE .. END

The basic conditional instruction is the IF...THEN...ELSE clause. This instruction has two forms:

```
IF expression THEN
        code block (executed when expression is true)
END
```

```
IF expression THEN
        code block (executed when expression is true)
ELSE
        code block (executed when expression is false)
END
```

Expression is any well-formed boolean expression.

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

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

7) Looping structures : 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.

- **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

- **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.

- **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
```

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
```

7. Functions

Functions generally requires you to provide them with data, and they return a value based on a specific operation on that data. If no argument requires, the parenthesis is omitted. Functions can be used in these cases.

Variable assignment:

```
Var_root = Sqrt(x)
```

Inside expresstion:

```
If len($my_str) > 12 Then
```

Arguments to a function:

```
Point #p1 = Trans(Sqrt(x), Sqrt(y))
```

Function	Argument	Explanation
Abs	Number x	Returns an absolute value A = Abs(-3.5) → A = 3.5
AOut	Number channel	Returns an analog output A = AOut(2) → Analog output value of channel 2
AIn	Number channel	Returns an analog input. A = AIn(2) → Analog input value of channel 2
Atan2	Number Y Number X	Arc tangent y/x as degree A = Atan(1, 1) → A = 45
Asc	String s [Number index = 1]	Returns ASCII value of indexed character of string A = Asc("sport", 2) → the second character 'p'
Bits	Number start_signal Number count	Continuous value of signals 1007 = on, 1006 = on, 1005 = off, 1004 =on A = Bits(1004, 4) → 1101(B) 13
Cos	Number X	Cos , X is degree A = Cos(90) → A = 0
CvtTrans	Point #joint	Convert joint location to trans location Point ptrans = CvtTrans(#pjoint)
Dest/#Dest		Target location of current motion instruction Point pold = Dest / Point #pold2 = #Dest
Distance	Position A	Distance from A to B

	Position B	A = Distance(p1, p2)
DX, DY, DZ	Trans p	Gets the component of X,Y,Z Xval = DX(pt1), Yval = DY(pt1), Zval = DZ(pt1);
Frame	Position porg Position px Position pxy Position pz	Calculate the coordinate frame Point newcoord = Frame(porg, px, pxy, pz)
Here/#Here		Returns current location Point pcur = Here Point #pcur_j = #Here
#Joint	Number j1 Number j2 …	Create joint location Point #pjnt = Joint(0, -20, 10) The default argument is 0 Point #porg = Joint(0) → Every joint location is 0
Len	String s	Length of string A = Len("sport") → A = 5
$Mid	String s Number index Number count	Gets the middle ranges of string $A = $Mid("sport", 2, 3) → A = "por"
Random		Random number is generated. The range is 0 to 1 A = Random
Round	Number x	Round of X A = Round(3.5) → A = 4
Rx, Ry, Rz	Number x	Gets the rotational transform. Point trx = Rx(30) → Rotate 30degree with x Axis
Sig	Number sig1 Number sig2 Number …	And operation of the given signals 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	Sine A = Sin(90) → A = 1
Sqrt	Number x	Square root A = Sqrt(4) → A = 2
Timer	Number id	Returns elapsed time A = Timer(1) → The current time value of Timer 1

#TouchJoint TouchTrans	Number index	Gets the touch probed location Point #A = #TouchJoint(1)
Translate	Position p	Translate the trans location Point A = Translate (P, 10, 2) → A is the translated location of P
	Number dx	
	Number dy	
	Number dz	
Trans	Number x	Create the trans location data or variable. 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 numeric A = Value("12") → A = 12

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

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

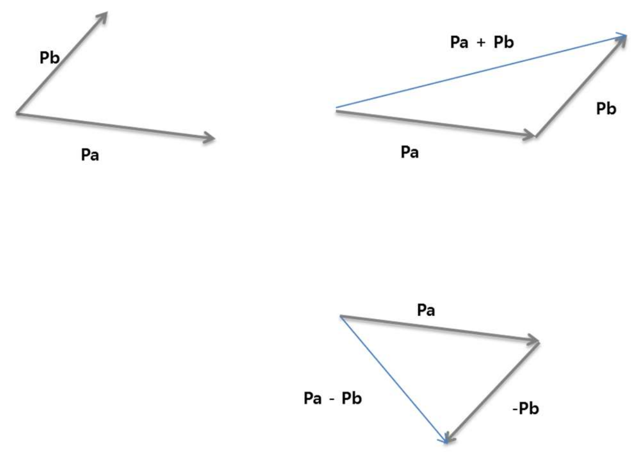


Figure 3 The operations of location variables

```
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. 현재 작업용 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.

2) Robot Motion

- MoveJ location → Joint interpolation motion. Abbreviation : MJ
- MoveL location → Straight line interpolation motion. Abbreviation : ML
- MoveC location1, location2 → Circular interpolation motion. Abbreviation : MC
- MoveX location, signal_no → Abbreviation : MX

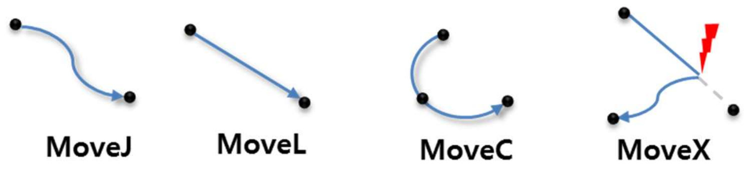


Figure 4 Robot Motion Instrunctions

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 #p1
SetDO 1
MoveJ #p2
SetDO 2
```

will cause external output signal #1 to be turned on immediately after the robot begins moving to #p1, rather than waiting for it to arrive at the location. When the second MoveJ instruction is encountered, CL waits until the motion to #p1 is completed. External output signal #2 is turned on just after the motion to #p.2 begins.

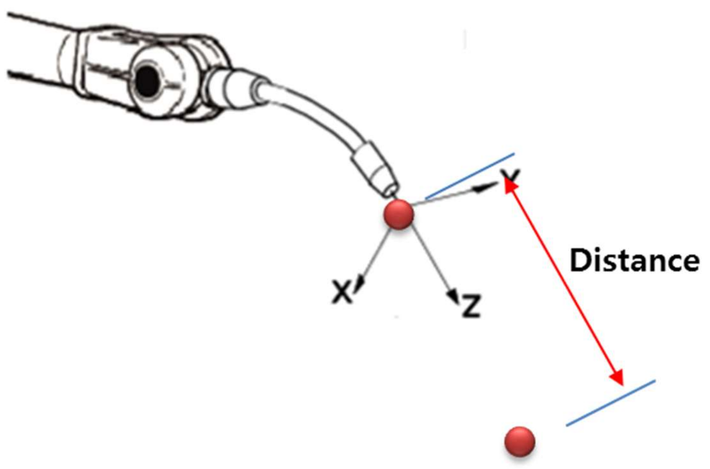
- Delay time
- Stable time

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

After approach and move to the target location, you can go back to the offset location with depart instruction

- 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]

Every motion instructions can have additional arguments. They are well conditioned digital output options and speed parameters.

  - 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.

- IncJ : Incremental joint move.
  - 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
```

3) Motion Modifiers

- 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 speed [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.

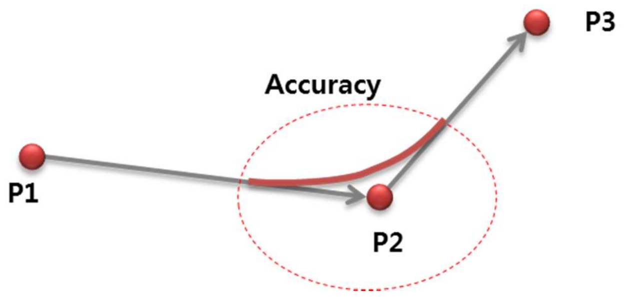


Figure 6 A continuous motion with accuracy

- 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        ;
    SetDO 1         ; Signal out immediately beginning motion toward P2
    MoveL p3
;prog2
    MoveL p1
    MoveL p2        ;
    Break
    SetDO 1         ; Signal out after reaching the target P2
    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.

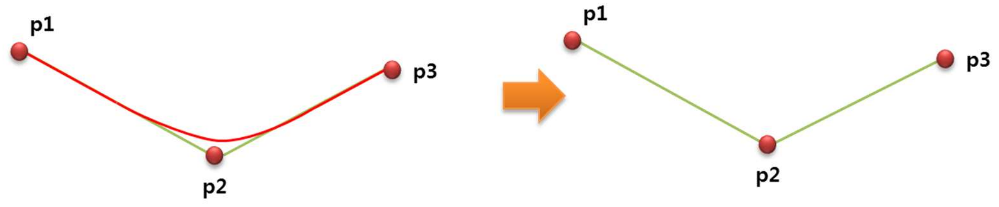


Figure 7 Path changes due to Break instruction

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

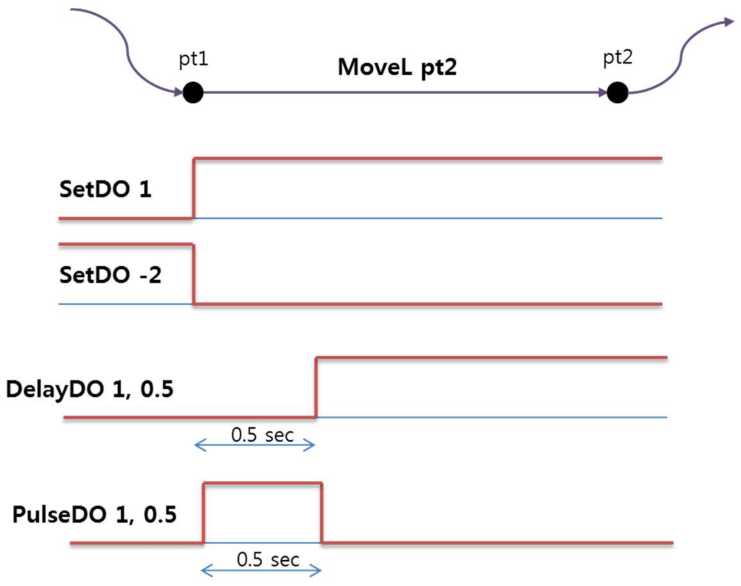
4) I/O

- 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.
- BsCondition index, type, start, end : Define the bundle signal.
  - ■ index : Bundle index. The range from 1 to 100.
  - ■ type : Signal condition type
    - 1 : Time, The value of start, end is second.
    - 2 : Distance, The value of start, end is distance.
  - ■ start : Departure condition. Ignore this condition for the negative value
  - ■ end : Arrival condition. Ignore this condition for the negative value.
- BsDO index, do1, do2, ... : Specify the digital output signal list for the given bundle condition

Following figure shows the timing chart for SetDO, DelayDO, PulseDO instructions moving toward pt2



You can define the complex condition by BsCondition, BsDo, such as before arriving pt2 ahead 5mm or 0.5sec

5) Touch Probe

EtherCAT servo driver provides special monitoring functions as touch probe. It detect the signal change and save the encoder position as fast as the driver can. For the details in its function, please refer to the ECAT driver manual.
CL provides the instructions to use touch probe function seamlessly.

- TouchEnable :
  - TouchEnable 1 : Enable the touch function.
- TouchStart watch_axis, signal_edge
  - watch_axis : Specify the axis list to monitor. To watch 1, 3 axes, set 5. LSB means Axis 1
  - signal_edge : Specify rising edge(1) or falling edge(0)
- TouchStop
- TouchWait : Wait until the touch point is detected.
- Functions getting touch poin
  - Point #p1 = #TouchJoint(1) ; touched point at the rising edge as a joint location.
  - Point pt1 = TouchTrans(1) : touched point at the rising edge as a trans location.
  - Point #p2 = #TouchJoint(-1) : touched point at the falling edge as a joint location.
  - Point pt2 = TouchTrans(-1) : touched point at the falling edge as a trans

location.

6) Network

CL provides the TCP/IP network instructions. It is similar to the standard socket programming. CL has two different modes on the network instructions. You have to choose the right mode if your application works as client or server.

- 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]

  Read and Write instruction may have a return code optionally. You can examine the return code and process the appropriate actions. In special case when TCPRead returns 0, it means the server went down.

- Server Mode
  - TCPSStart socket_variable, port number
    - If socket variable is negative, the error has occurred.
  - TCPSStop socket_variable
  - TCPSAccept soket_variable client_socket_variable
  - TCPSCClose socket_variable : close client
  - TCPSRead client_socket, string variable, [return code]
  - TCPSWrite client_socket, string variable, [return code]

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]
```

7) Conveyor Tracking

The details about conveyor tracking programming are shown in the conveyor tracking manual. This section introduces the instructions that CL provides for the conveyor tracking.

- TkSetSig cvid, trigger signal #

Setup trigger signal to add object into the object queue
This can be defined in the Conveyor Settings UI. It may not need in the program.

- TkObjWait cvid, [timeout], [result]

Wait until the object comes in the range of working.

- 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 :

마지막에 삽입된 object 의 정보를 얻어 냅니다.

- 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.**

<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) ETC

- ULIMIT axis1, axis2, …
- LLIMIT axis1, axis2, …

The upper limit and lower limit are defined programmatically.

- TIMER

CL has timer to measure the elapsed time. You may measure the application cycle time.

  - SetTimer timer#, time : Set timer value as seconds
  - Timer(timer#) : Read elapsed time

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

9) User coordinate instructions

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 Instructions

- 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
  - 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

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.

Instructions :

EPSmode On or EPSmode Off

Enable or disable the external program selection mode 결정합니다.

EPSWait

Wait signal and jumps selected program

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.

1) Example : Program selection with EPS MODE

- EPS main program : EPSGO

```
HOME
EPSMode ON
EPSWait
```

- Program name : Pg1

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

- Program name: Pg2

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

- 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

2) Program selection with Bits instruction and SWITCH CASE structure

The same program selection can be accomplished Bits instruction and Switch Case structures.

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

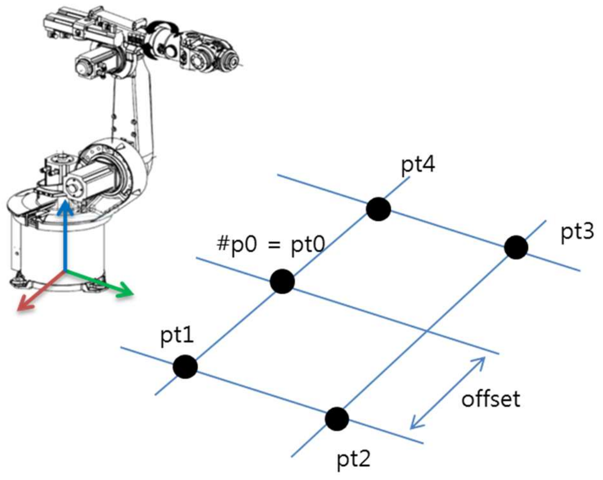
11. Example programs

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 distance

Point pt0 = CvtTrans(#p0)         ;   Convert Joint location to Trans location

Point pt1 = Translate(pt0, offset, 0, 0) ; Define corner location using Translate

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
```

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 pt3

MoveL 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.

3) IO Instructions

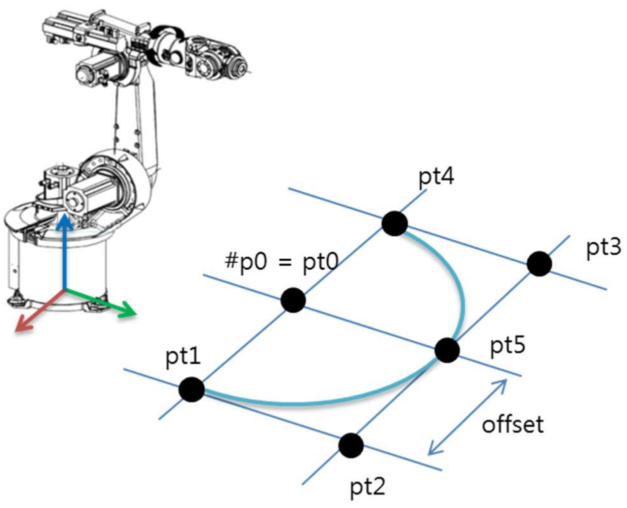
This examples shows how to use Digital Input and Digital output signal.

```
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      ; Digital output signal is On on departing to pt3.

MoveL pt3

SetDO -3, -5    ; Output signal is off

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.

4) MoveC

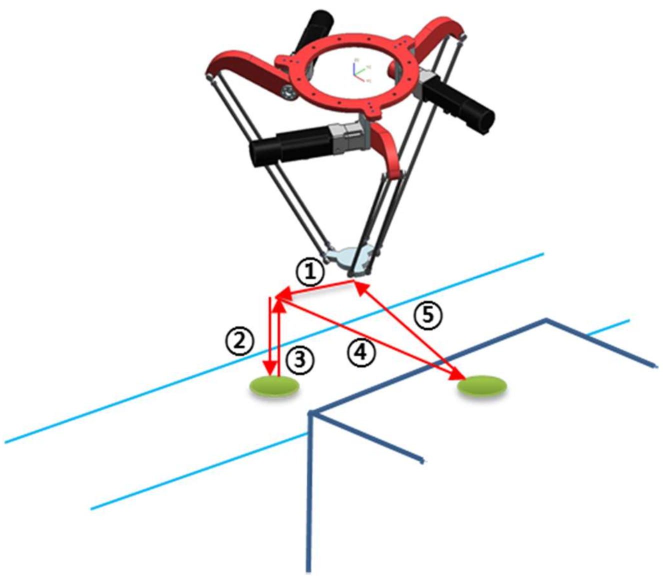
This example shows how to move robot following a circular path. To define a circular movement, two locations must be defined.



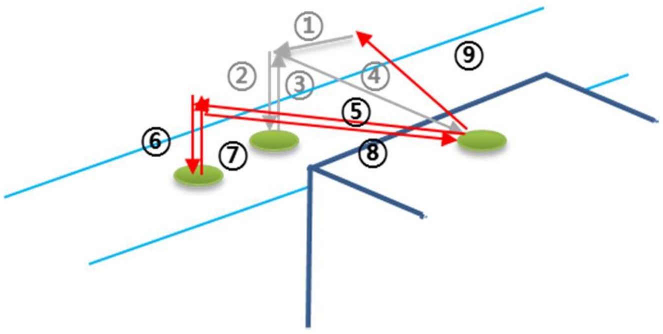
```
MoveJ #p0

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
```

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.