CL Programing Manual

Revised on 2018. 03. 21.

Printed in the Republic Of Korea - 2016년 10월. 초판

CL Programing Manual Revision History

Revision	Data	Comment	비고
1.01	16.10.11	최초 작성	
1.02	18.03.21	명령어 설명 추가	

목차

1. 개요

CL(Core Language) 로봇 언어는 다양한 상황의 로봇 프로그램이 가능하도록 고수준의 언어 사양을 제공합니다. 프로그램의 제어 흐름을 설정할 수 있는 조건문, 반복문이 가능하고 상수, 변수 등을 이용하여, 프로그램의 재사용성을 향상시켰습니다.

CL 언어 설명서인 본 매뉴얼을 읽기 전에, coreCon 사용자 설명서를 통해 coreCon에서 로봇의 작동 및 프로그램 작성에 관한 순서를 참고하기 바랍니다.

2. Data & Variables

CL 에서 사용하는 Data 형식은 현재 숫자, 각축위치변수, 직교위치변수, 그리고 문자열 데이터로 네 종류로 구성되어 있습니다. 이러한 기본 Data 는 배열 형식으로 사용할 수 있습니다. 또한, Data 는 함수나, 명령어의 인자로 사용될 수 있고 변수를 이용하여, 데이터를 저장할 수 있습니다.

데이터를 저장하는 변수는 문자와 숫자의 조합으로 정의할 수 있습니다. 그러나, 숫자로 시작할 수 없고, 변수 가운데 특수 문자를 사용할 수 없습니다. 예를 들어 V100 은 가능하나, 100V 는 불가능합니다. 또한 V@100 과 같이 특수 문자가 와도 불가능합니다. 알파벳 문자와 숫자 외에 변수로 사용할 수 있는 문자는 '.', '_'입니다. 즉, V.temp, V_press 등은 변수로 사용이 가능합니다. If, random 등과 같이 이미 다른 의미로 사용되고 있는 예약어 역시 변수로 사용 할 수 없습니다.

또한, 변수를 구분하기 위해 문자열 데이터와 Joint 위치 변수는 각각 변수 앞에 $와 #을 추가하여 사용해야 합니다.

- Macro program 의 길이는 19 자, 변수(variable)는 15 자가 제한입니다.

1) 숫자(Number)

숫자 값은 숫자 값을 반환하는 숫자, 변수, 연산자 및 함수의 조합입니다. 숫자 표현식은 수학적 계산뿐 아니라 프로그램 명령어의 인자 값으로도 사용됩니다. CL 에서 사용되는 숫자 변수는 다음 세가지로 나누어집니다.

INTEGERS : 정수
정수는 분수부분이 없는 값입니다.(전체 자릿수). 정수의 범위는 -16,777,216 에서 +16,777,216 까지 이다. 이 범위를 초과하는 값은 값은 7 개의 유효 자릿수로 반올림됩니다. 정수 값은 대개 10 진수로 입력되어지지만 2 진수나 16 진수로 입력하는 것이 더 편리할 때도 있습니다. 16 진수 값은 ^H 가 숫자 앞에 추가 되어 설정됩니다.

REAL NUMBERS : 실수
실수는 -3.4E +38 ~ 3.4E +38 범위의 정수 부분과 분수 부분을 둘 다 가지고 있습니다. 실수는 정수와 같이 양수, 0, 음수로 되어 있으며, 과학적인 표기법으로 나타내질 수 있습니다. 실수는 실제 값은 약 7 자리의 정확도로 저장되지만 실제 값은 계산 오류로 인해 정밀도가 낮을 수 있습니다.

LOGICAL VALUES : 논리적 값
논리적 값은 오로지 ON 과 OFF 두 개의 상태로 이루어져 있습니다. 이 두가지 상태는 각각 TRUE 와 FALSE 로도 불립니다. -1 은 ON 또는 TRUE 상태일 때, 0 은 FALSE 또는 OFF 상태일 때 할당됩니다.

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

현재 CL 에선 정수형과 실수형을 구별하지 않고, 실수형만 사용하고 있습니다. 정수형은 실수형에서 소수점 자리를 제거한 숫자입니다.

1) 문자열 데이터(String)

문자열 데이터는 ""로 둘러싸인 데이터 입니다. 문자열 변수는 다른 변수와 구별하기 위해 $로 시작합니다.

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

문자열 간의 합 가능함

```
$test1 = "K"
$test2 = "im"
$test3 = $test1 + $test2 = "Kim" (숫자 변수와 합은 불가)
```

2) 위치 데이터(Joint)

각 축의 Joint 위치 값을 저장하는 위치 데이터입니다. 직교위치데이터는 다양한 형태의 로봇 자세가 나올 수 있으나, Joint 위치 데이터는 정확한 자세 한가지만을 표현하게 됩니다.

Joint 위치의 값은 개별 로봇 Joint 의 정확한 위치로 표시됩니다. Joint 위치는 고려해야 할 몇 가지 특징이 있습니다. 그리고 이러한 특성은 기록된 조인트 각도에서 비롯됩니다.

Joint 위치의 장점 : 재생정밀도가 달성되었으며, 위치에서 로봇 configuration 에 대한 모호성이 없습니다.

Joint 위치의 단점 : 기록된 값은 어느 로봇 모델이든지 사용할 수 있지만, 로봇의 tool center point 위치는 다른 물리적 크기의 로봇에 의해 사용될 때마다 다르다. 로봇 작업공간에서 위치

변경을 보상하기 위해 정밀한 위치를 쉽게 수정할 수 없습니다. 왜냐하면 이러한 변경은 작업공간에서의 위치와 모든 로봇의 Joint 위치 사이의 완벽한 관계를 요구하기 때문입니다.

Joint 위치 데이터는 직교 위치데이터와 구별하기 위해 변수 이름을 #으로 시작해야 합니다. 위치데이터는 Teaching 을 통해 현재 위치를 저장할 수 있고, Joint() 라는 함수를 사용하여 사용자가 원하는 임의의 값으로 정의 할 수 있습니다.

```
MoveJ #p1 ; #p1 으로 정의된 위치로 이동
MoveL #p2 ; #p2 로 정의된 위치로 직선 이동

Point #p3 = Joint(10, 20) ; #p3 의 위치 변수 정의
Point #p4 = #p3 ; #p3의 위치 값을 #p4에 복사

delta = 5
Point #p5 = Joint(delta, delta*2, -delta*2) ; 숫자 변수를 이용한 위치 값 정의
```

위의 예에서와 같이 위치 변수의 정의는 항상 Point 위치변수 = 목표 값 형식으로 정의해야 합니다. Joint() 함수를 이용한 #p3 = (10,20, 0, 0, 0, 0) 이 됩니다. 즉, 입력되지 않은 값은 0 으로 설정이 됩니다.

3) 직교 위치 데이터(Transformation)

Transformation 위치 데이터로 직교 좌표계 상에서, 자세와 위치를 포함하는 Homogeneous Transform Matrix 형식으로 내부에 저장됩니다. transformation 위치 데이터는 로봇의 Base 에 고정된 프레임에 참고된 Cartesian(XYZ)으로 위치가 정의됩니다. tool center point 의 위치는 X, Y, Z 좌표로 정의되며, 툴의 원점은 좌표축에서 측정한 세 개의 Euler Angle 로 정의됩니다. Euler Angle 은 (A, B, C)로 표현되며, joint 위치변수와의 차이를 구별하기 위해 Trans 위치 변수 앞에는 따로 정의된 문자가 없습니다. (joint 위치 변수 앞에는 #이 추가되어 사용됩니다.)

transformation 위치의 장점 : 하나의 로봇에서 사용을 위해 정의된 값은 작업공간 좌표의 측면에서 정의되었기 때문에 유사한 작업 영역을 가진 다른 로봇에서 사용할 수 있습니다. Transformations 는 로봇 작업공간 내에서 위치를 쉽게 바꿀 수 있습니다. transformation 위치의 가장 큰 장점은 위치를 값의 조합으로 위치를 정의할 수 있다는 것입니다. 이것을 그의 이름은 복합 혹은 상대적인 transformation 이라고 부릅니다. 이러한 값은 이거의 fixturing 에 관한 위치를 정의하는 데 사용됩니다.

transformation 위치의 단점 : transformation 위치는 작업공간에서의 좌표에서 tool center

point 의 위치를 정의하므로, 해당 위치에서 특정 로봇 configuration 에 대한 정보가 제공되지 않습니다. transformation 이 로봇 동작의 도착지를 정의하는데 사용될 때마다, coreCon 은 동등한 정밀 위치로 transformation 위치를 변환해서 개별적인 joint 가 음직이는 방법을 알아야 합니다. 그리고 이 변환 작업은 작은 위치 오류를 일으킬 수 있습니다. 이러한 단점에도 불구하고, transformation 위치는 일반적으로 정밀 위치보다 훨씬 더 편리하게 사용됩니다.

```
MoveJ pt1 ; pt1 으로 정의된 위치로 이동
MoveL pt2 ; pt2 로 정의된 위치로 직선 이동

Point pt3 = Trans(10, 20) ; pt3의 위치 변수 정의
Point pt4 = pt3 ; pt3 의 위치 값을 pt4에 복사

delta = 5
Point pt5 = Trans(delta, delta*2, -delta*2) ; 숫자 변수를 이용한 위치 값 정의
```

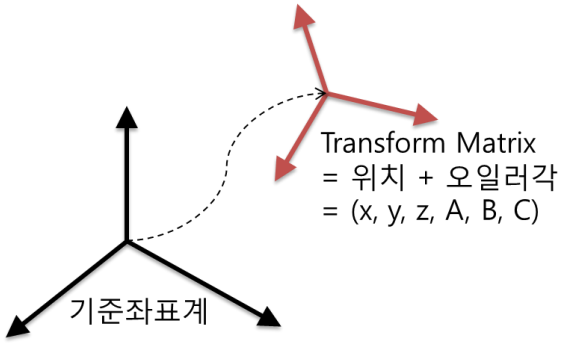


Figure 1 직교 위치데이터의 정의

직교 위치데이터는 기준 좌표계에 의해 정의되기 때문에, 기준 좌표계에 따라서 실제의 위치가 바뀌게 됩니다. 따라서, 기준 좌표계를 변경할 시에는 로봇의 자세가 변경될 수 있는 것에 주의하여야 합니다.

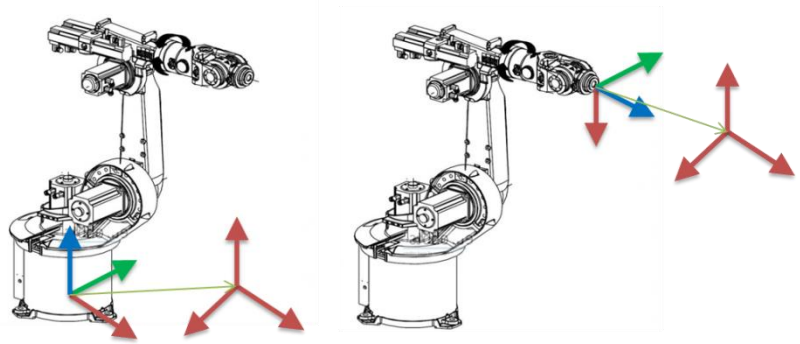


Figure 2 기준 좌표계에 따른 Trans 위치

위의 그림은 Trans(100, 100, 0)으로 정의된 직교위치 변수의 위치가 기준 좌표계에 따라서, 전혀 다른 곳을 표현할 수 있음을 보여 주고 있습니다. 이와는 달리 Joint 위치 변수는 로봇 관절의 각을 직접 정의하였기 때문에, 항상 동일한 위치를 지시하게 됩니다.

4) 배열(Array)

배열은 연속된 데이터를 인덱스를 이용하여 사용할 수 있도록 한 데이터로서 단일 이름을 공유하는 값의 그룹입니다. 위치 변수는 scalar 또는 배열일 수 있습니다. 위치 scalar 는 단일 위치 값입니다. 배열의 각 값은 배열의 요소라고 합니다. 위치 배열의 요소는 배열 이름에 괄호 안에 인덱스가 포함된 형태로 숫자 배열의 요소와 정확하게 같은 방식으로 명시됩니다. 예를 들어, "part[7]"은 "part"라는 배열의 7 개의 요소를 나타냅니다. 인덱스는 0~9999 까지의 범위의 정수여야 합니다. . CL 에서 배열은 1 차원 배열만 지원합니다. 네 가지 예시를 다음과 같이 나타내었습니다.

```
A[1] = 10 ; 숫자 배열
A[2] = 20
$name[0] = "John" ; 문자열 배열
Point #p1[0] = Joint(10,0,3) ; 위치데이터 배열
```

5) Local Variable 및 Subroutine Parameter

1. Subroutine Parameter 정의
   프로그램 맨 처음에, 아래와 같이 .PARAM Argument로 사용할 변수를 정의합니다.

프로그램 1: main(Macro Program name: main)

```
.PARAM $retstr
.LOCAL $strval
$retstr = ""
$strval = "Core"
Call prog($strval, $retstr) ;prog()는 sub 프로그램을 call함
```

프로그램 2: prog (Macro Program name: prog)

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

2. Local Variable 설정

Subroutine 내에서만 유효한 변수를 정의하고자 한다면, .LOCAL Argument명령을 사용하여 변수를 정의할 수 있습니다.

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

변수 호출 시 항상 LOCAL 변수를 먼저 찾기 때문에, PARAM 변수와 동일한 이름의 변수가 있을 시 PARAM 변수는 사용되지 않습니다.

3. Subroutine과 Local Variable 관계

Subroutine Parameter도 LOCAL 변수로 사용되기 때문에, LOCAL 변수와 동일한 속성을 가집니다. 하지만, Parameter로 입력된 변수는 Reference로 넘어가기 때문에, Subroutine에서 변경된 데이터가 입력된 변수데이터로 변경이 됩니다. 위의 예제에서 prog에 해당하는 $strvar1의 내용을 변경할 경우 프로그램 2번 prog의 $str이 변하게 됩니다.

4. 변수 초기화

```
.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
```

6) 단위

특별한 설정이 없는 한 기본적인 단위는 다음과 같습니다.

- 길이(Distance) : mm
- 각도(Degree) : deg
  - 각도는 Degree 로 표현되기 때문에 Sin(30) 과 같은 함수에 사용될 각도는 Radian 이 아닌 각도입니다.
- 속도, 가속도(Velocity) : %
  - 최고 속도 및 최대 가속도에 대한 비율을 백분위로 표현합니다.
  - 최대 속도는 로봇의 설정에 정의가 되어 있습니다. 예를 들어 로봇 설정에서 최대 속도가 300mm/s 라면, speed 값을 10 으로 설정할 경우, 300 * 10 %인 30mm/s 로 설정됩니다.

3. Constant

CL 언어에서는 몇 가지 상수가 정의되어 있습니다. 특정 명령어에 이러한 상수를 사용함으로써 의미를 분명히 할 수 있습니다.

- 보조 상수 : 프로그램 언어에 사용되는 단위
  - 길이(Distance) : mm
  - 시간(Time) : s
  - 속도, 가속도(Velocity) : mm/s, sec, mm/min
- on / off : ON, OFF
- true/false : TRUE, FALSE
  - 내부적으로 ON/TRUE 의 값은 -1 이며, OFF/FALSE 의 값은 0 으로 설정되어 있습니다.
- PI : 3.141592... = 원주율
- NULL : identity trans matrix
- Icon 정의 상수
  - ICONE : Error Icon = 0
  - ICONW : Warning Icon = 1
  - ICONI : Information Icon = 2
  - TPWrite 명령어에서 사용할 수 있으며, Message 창에 지정된 Icon 이 표현되게 됩니다.

```
Speed 80 mm/s ; 80 mm/s의 직접 속도 지정
Speed 5 sec ; 구간 거리를 5초간 이동
WaitTime 1.5 s ; 1.5초간 대기
A = ON ; A의 값은 내부적으로 -1이 됨
B = OFF ; B의 값은 내부적으로 0이 됨
```

4. Operator

1) Unary Real Operators

- **COM** : 보수(Complement)
- **-** : 음수
- **NOT**

```
A = -3
Aa = -A ; Aa는 정수로 3, Hex로 0x00000003 이 됩니다.

B = COM Aa ; Aa = 0xFFFFFFFC 가 됩니다.

C = TRUE
D = NOT C ; D = FALSE가 됩니다.
```

2) Binary Real Operators

- **^** : 제곱
- ***** , **/** , **MOD** : 곱셈 및 나눗셈(몫), 나머지
- **+, -** : 덧셈 및 뺄셈
- **<**, **<=**, **=<**, **==**, **>=**, **=>**, **>**, **<>** : 비교연산자
- **BAND**, **BOR**, **BXOR** : Bit 별 연산자
- **AND**, **OR**, **XOR** : 논리연산자

```
a = x^4 + 3 * (3 – x) ; 기본연산자
if a AND b then ... ; 논리연산자로서 a와 b를 모두 만족해야 조건 실행
if a <= b then ... ; 비교연산자
if (a < b) AND (c > d) then ... ; 비교연산자 + 논리연산자
Wait Sig(1001) OR Sig(1002) ; 논리연산자
```

5. Functions

Function 은 return 값을 갖는 명령어로, return_value = function(args...) 의 형식으로 사용됩니다. Argument 가 없을 경우, ()를 생략하고, function 만 기입합니다.
인자 리스트 중 []로 표시된 인자는 옵션 인자로 생략될 경우, 기본 값이 입력되게 됩니다.

아래 표는 제공하는 Function 리스트로, 자세한 설명은 참조 설명 부분을 참고하기 바랍니다.

함수명	인자	설명
Abs	Number x	인자 값의 절대값이 설정됩니다. A = Abs(-3.5) → A = 3.5
AOut	Number channel	출력 analog 값을 return 합니다. A = AOut(2) → 2 번 채널의 analog output 값
AIn	Number channel	입력 Analog 값을 return 합니다. A = AIn(2) → 2 번 채널의 analog input 값
Atan2	Number Y Number X	Arc tangent y/x 의 값을 return 합니다. 결과 값은 Degree 로 나옵니다. A = Atan(1, 1) → A = 45
Asc	String s [Number index = 1]	문자열 s 의 index 에 해당하는 문자의 ascii 값을 return 합니다. A = Asc("sport", 2) → "sport"의 두 번째 글자 'p'의 ascii 값을 할당합니다. 옵션 인자 기본값은 첫 번째 글자로서 입력하지 않으면 1 이 설정됩니다.
Bits	Number start_signal Number count	시작 Digital 신호 번호로부터 연속된 개수의 시그널 값을 읽어 들여 숫자화 합니다. 1007 = on, 1006 = on, 1005 = off, 1004 =on 일경우, A = Bits(1004, 4) → 1101(B) 즉, 13 을 할당합니다.
CheckSum	Strings, Type	Serial 통신에서 통신 데이터의 유효성을 검사합니다. chkvalue = CheckSum($data, type) Type: 0: CRC16 1: VRC(수평 XOR) 2: LRC(수직 XOR) 3: SUM(sum) 4: OR(or)
$Chr	Hex hex_code	Hex code 를 Ascii Char 로 변환합니다. hex_code = ^h31

		$char = $Chr(hex_code) ; '1'
Cos	Number X	Cos 값을 할당하며, X 의 값은 degree 입니다. A = Cos(90) → A = 0
CvtTrans	Point #joint	Joint Point 값을 사용된 Tool 과 Base 좌표계를 기준으로 Trans Point 값으로 변환합니다. Point ptrans = CvtTrans(#pjoint)
Dest/#Dest		현재 계획된 모션의 도착점을 각각 Trans/Joint 위치 변수로 할당합니다. Point pold = Dest / Point #pold2 = #Dest
Distance	Position A Position B	A 와 B 사이의 거리를 구합니다. A = Distance(p1, p2)
DX, DY, DZ	Trans p	각각 직교 위치 좌표 p 의 X, Y, Z 값을 구합니다. Xval = DX(pt1), Yval = DY(pt1), Zval = DZ(pt1);
ErrorCode	ErrorCode	Error code 를 반환합니다. a = ErrorCode
Frame	Position porg Position px Position pxy Position pz	좌표계 frame 을 구합니다. 만들어지는 좌표계의 원점은 porg 이고, x 방향 벡터는 px – porg 가 됩니다. Pxy 점으로 y 축이 존재하는 평면을 정의하고, pz 로 z 축 방향을 결정합니다. Point newcoord = Frame(porg, px, pxy, pz)
$HexStr	String s	Ascii char 를 hex 로 변환하여 String 으로 반환합니다. $char = "12" $hex_str = $HexStr($char) ; "3132"
Here/#Here		현재 위치를 할당합니다. Point pcur = Here Point #pcur_j = #Here
#Joint	Number j1 Number j2 …	각 축의 joint 값을 가지는 Joint 위치 변수를 생성시킵니다. Point #pjnt = Joint(0, -20, 10) 생략된 인자는 모두 0 이 할당됩니다. Point #porg = Joint(0) → 모든 Joint 가 0 인 점 할당
$Int16B	Hex hex_code	Hex_code 를 2 자리 Ascii char 로 변환합니다 hex_code = ^h3161 $hex_str = $Int16B(hex_code) ; "1a"
Len	String s	문자열 s 의 길이를 구합니다. A = Len("sport") → A = 5
$Mid	String s	문자열 s 의 index 에 해당하는 위치부터 개수만큼의

	Number index Number count	문자열을 구합니다. $A = $Mid("sport", 2, 3) → A = "por"
Random		Random 변수를 생성시킨다. 범위는 0 ~ 1 이다. A = Random → A = 임의의 값
Round	Number x	X 를 반올림 시킨 값을 구한다. A = Round(3.5) → A = 4
Rx, Ry, Rz	Number x	각각 x 각도만큼 회전한 회전 Trans matrix 를 return 합니다. Point trx = Rx(30) → x 축을 30 만큼 회전하는 Trans matrix
Sig	Number sig1 Number sig2 Number ...	Sig1, Sig2, ... 등과 같이 입력된 모든 signal 의 and 결과를 return 합니다. 1002 = on, 1003 = off 일 경우 A = Sig(1002, 1003) → A = 0(FALSE) 1002 = on, 1003 = on 일 경우 A = Sig(1002, 1003) → A = -1(TRUE)
Sin	Number x	Sin 값을 할당하며, x 는 degree 입니다. A = Sin(90) → A = 1
Sqrt	Number x	Square root 값을 구합니다. A = Sqrt(4) → A = 2
Timer	Number id	Id 에 해당하는 Timer 값을 return 합니다. A = Timer(1) → 1 번 Timer 의 값을 return
#TouchJoint TouchTrans	Number index	Touch 된 Signal 의 상태가 Index 에 해당될 때 위치 값을 받아옵니다. Point #A = #TouchJoint(1) → A 는 외부 Signal 이 Rising 될 때 현재 위치를 가지게 됩니다.
Translate	Position p Number dx Number dy Number dz	P 점을 dx dy, dz 만큼 이동한 Trans matrix 를 구합니다. 입력되지 않은 이동양은 0 으로 설정됩니다. Point A = Translate (P, 10, 2) → A 는 P 점을 x 축으로 10 만큼, y 축으로 2 만큼, z 축으로 0 만큼 이동한 값이 들어갑니다.
Trans	Number x Number y Number z Number A	x, y, z, a, b, c trans point 의 값을 직접 입력하여, Trans 위치 변수를 생성시키는 함수입니다. Point p = Trans(0, -20, 10, 20, 20, 10)

	Number B	
	Number C	
Value	String s	S 를 숫자로 변환하여 그 값을 할당합니다. A = Value("12") → A = 12

6. Comment : ;

주석문은 " ; "로 시작하는 문장입니다. 실행 시 무시되는 문장입니다.

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

7. Controlling the program flow

프로그램의 동작을 조건에 따라 조정할 수 있는 제어문에 대한 설명입니다. CL 에서 제공하는 제어문은 아래와 같습니다. 프로그램 도중 명령 실행을 중단하거나, 대기하는 제어문도 사용할 수 있습니다, 제어문은 1 loop 당 0.002mmsec 의 delay 을 갖습니다.

- **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 문

GOTO 명령어는 프로그램 label 명령어가 있는 곳으로 분기하여 즉시 해당 프로그램을 실행합니다.

- **GOTO** label [IF condition] : condition 이 TRUE 일 경우, label 로 분기합니다.
- **IF** condition **GOTO** label : condition 이 TRUE 일 경우, label 로 분기합니다.

label 은 프로그램 코드의 시작 부분에 입력된 정수입니다. label 은 프로그램 스텝 번호와 같지 않습니다 : 스텝 번호는 시스템에 의해 할당되는 번호이고, label 과는 무관합니다. label 은 코드의 시작 부분에 프로그래머에 의해 작성됩니다.

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

간단한 조건일 경우, 아래와 같이 IF와 동시에 사용할 수 있습니다.

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

2) 다른 프로그램 호출

- **CALL** subroutine

CALL 명령어는 서브루틴 호출을 구현하기 위해 사용되고, 다른 프로그램을 호출 합니다. CALL 명령어는 기존의 프로그램을 중지하고, 새로운 프로그램을 실행시킵니다. CALL 명령어 후에 실행된 새로운 프로그램의 수행이 끝나면, 기존의 프로그램을 다시 실행시킵니다.

CL 에서는 프로그램 파일 하나하나가 서브 프로그램이 됩니다. 즉, Call 에서 사용할 subroutine 이름은 별도로 작성한 프로그램 이름이 됩니다. 즉, CALL 명령어의 인자 값으로 사용하기 위해 다른 프로그램이 존재하고 있어야 합니다.

아래 예는 Prog1, Prog2, Prog3 세 개의 프로그램을 작성하여, Prog3 이 조건에 따라, Prog1, Prog2 를 각각 호출하는 예제 입니다.

- Call subroutine(arg1,arg2)
  Call 명령으로 다른 프로그램 호출 시 Argument 사용이 가능합니다.
- 주의 사항
  각 Task 에서 동시에 동일한 서브루틴 호출은 금지되어 있습니다.

```
; 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
- **Ignore Signal# ; 해당 signal interrupt 종료**

CL 은 인터럽트 핸들러를 제공합니다. 디지털 입력 신호의 상태 전환에 따라 프로그램을 중단할 수 있습니다.

특정 입력 신호상태가 변경될 경우, 로봇 동작은 즉시 중지되며, 지정된 프로그램을 실행한 후, 원래의 프로그램으로 돌아오게 됩니다.

로봇이 동작 중일 때는 동작을 즉시 멈추고, 다른 프로그램을 실행 시킬 수도 있고, 동작을 완료한 다음 다른 프로그램을 실행 시킬 수 있습니다.

반복하여 지정된 프로그램을 실행하기 위해서는 입력 신호의 상태 전환이 필요합니다.

```
; Prog1
        MoveJ #pdrop
        SetDO hand1
        Interrupt 1002, Prog2 ; 1002가 On이 될 경우 로봇의 동작이 멈추고, Prog2
가 실행됩니다.
; Prog2   Interrupt Handler
        MoveJ #ppick
        SetDO hand2
```

4) 프로그램 실행 탈출 : Return, Stop, Pause

- **RETURN** : 현재 실행중인 프로그램 동작을 중지하고, 상위의 프로그램으로 돌아

갑니다.

- **STOP** : 현재 실행중인 프로그램 동작을 중지하고, 최 상위의 메인 프로그램으로 돌아 갑니다.
- **Pause** : 로봇 실행을 중지하고, Hold 상태가 됩니다. 프로그램은 Run operation 으로 재개될 수 있습니다.

아래 예를 살펴 보면, ProgMain 이 Main 프로그램으로 실행할 경우, 입력 신호 1001 의 상태에 따라, Prog1 이 호출되거나, Prog2 가 호출 됩니다. Prog1 이 호출 될 경우, 조건에 따라서, Prog2 가 호출되는데, Prog2 가 호출 되면, STOP 이 실행되기 때문에, 항상 ProgMain 으로 돌아가게 됩니다.

```
; 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 ; 상위프로그램으로 돌아갑니다.
             END
        END
        Pause ; 로봇은 정지하고, 사용자의 Run명령으로 다시 실행을 재개합니다.
; Prog2
        MoveJ #ppick
        SetDO hand2
        Stop ; ProgMain으로 돌아갑니다.
```

5) 대기

- **WAIT** condition, [Timeout], [Result]

조건이 만족 될 때까지 대기합니다. Timeout, Result 옵션을 입력하게 되면, 무한정 대기가 아닌, Time out 시간 까지만, 대기하고, 그 때의 결과를 Result 에 담아 둡니다.

즉, Timeout 옵션을 사용할 경우, Result 결과를 받아, 조건이 만족되었는지 아니면, Time out 으로 대기를 중지한 것인지를 구별할 수 있습니다.

```
        Wait 1001
; 1001이 on이 될 때까지 무한정 대기
        Wait Sig(-1001, 1002), 2, result
; 1001이 off가 되고, 1002가 on이 될 때까지 2초간만 대기한 후,
; 2초 뒤에 Sig(-1001, 1002)의 결과가 result로 입력됩니다.
        If result then ;
            TPWrite ICONI, "Job succedded"
        Else
            TPWrite ICONI, "Timeout"
        End
```

참고. SIG(1001, -1003) : 1001 이 ON 이고, 1003 이 OFF 가 될 때까지 대기. 즉 AND 조건이 됩니다. OR 조건은 Wait Sig(1001) OR Sig(-1003) 과 같이 입력합니다.

또한, Wait 에 쓰이는 조건문은 아래와 같이 Id 가 1 인 Timer 값이 100 초가 넘었을 경우, 숫자 변수 n 의 값이 100 이 넘었을 경우와 같이 사용 가능합니다

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

- **WaitTime** duration_second

주어진 시간 만큼 대기합니다. 시간 입력은 초단위가 됩니다.

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

6) IF 문

IF 문 구조는 아래와 같이 IF – 조건절 – 실행문 – ELSE – 실행문 – END 의 형태로 구현됩니다. 각각의 명령어는 한 줄씩 위치되어야 합니다.

```
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) 반복문 While ... Do, Do ... Until, For

반목문은 While 문과 Do 문이 있으며, Loop 변수를 사용하고자 할 때는 For 문을 사용합니다.

- **While** condition **Do** : 조건을 체크한 후, Do 문장을 실행합니다.
- **Do** ... **Until** condition : 적어도 1 회는 Do 문장이 실행되고 난 후, Until 이후의 조건을 체크하게 됩니다.
- **FOR** loop_variable = initial **TO** last **STEP** increment : STEP 생략 시 기본 1 증가

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

아래 프로그램은 For 문을 이용한, Palletizing 에 유용한 격자의 공간을 프로그램으로 움직이게 한 예제 입니다.

```
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)　　선택문 SWITCH … CASE　DEFAULT　END

여러 개의 선택으로부터 특정 선택을 수행할 경우, Switch 문을 사용합니다.
Switch 문의 문법은

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

END
```

입니다.

```
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

Instruction 은 Function 과 달리 Return 값이 없이, 로봇의 명령어로 실행됩니다. 일부 명령어의 상태를 반환하는 Instruction 들은 변수를 인자로 전달하여, 그 변수에 결과값을 담아 올 수 있습니다. Instruction 에는 로봇의 주요 동작 명령어, IO 설정, 대기, 통신등과 관련한 명령이 있습니다. 각각의 상세한 명령어 설명은 후반부의 참조 설명서를 참고하기 바랍니다.

명령어를 설명하기 전에, 우선 CL 에서 사용하는 위치 변수의 연산에 대해 알아 봅니다. 위치 변수 연산을 사용하게 되면, 위치 관련 로봇 프로그램에서 고수준의 프로그램을 만들 수 있습니다.

1) Transform & Position

동작 제어 명령어를 들어가기 전에, transformation 과 transformation 연산에 대한 개념을 파악해야 합니다. CL 의 trans 위치 변수는 이전에 언급한대로 homogeneous transformation 입니다. homogeneous transformation 에서 가장 중요한 연산은 위치 이동 또는 회전을 의미하므로 matrix 곱셈을 하는 것입니다. CL 은 연산자 +와 -로 forward transformation 과 inverse transformation 을 제공합니다.

Trans 변수 연산자 : + / -

Trans 위치 변수는 homogeneous transform matrix 입니다. 내부적으로는 4X4 매트릭스이지만, (X, Y, Z, A, B, C)로 표시됩니다. (A, B, C)는 회전을 나타내는 Euler angle 입니다. 따라서 연산자 +는 매트릭스 곱셈을 의미하며, 연산자 -는 역행렬의 곱셈을 의미합니다.
예를 들어, trans 위치 변수 Pa,Pb, 두 연산은 다음과 같습니다.

Pc = Pa + Pb → Pc = Matrix Pa * Matrix Pb
Pc = Pa – Pb → Pc = Matrix Pa * Matrix Pb^-1

Trans 위치 변수 Pa, Pb 가 있을 때, Pc = Pa + Pb 연산은 Trans Matrix 의 Multiplication 이 됩니다. 또한, Pc = Pa – Pb 연산은 Pa 에 Pb 의 Inverse Trans matrix 를 곱한 결과가 됩니다.

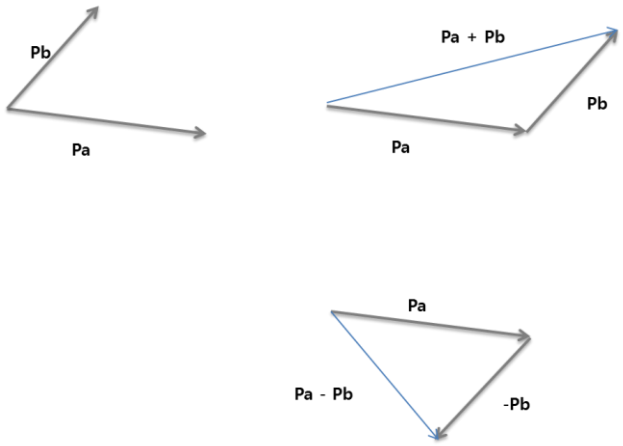


Figure 3 위치변수의 연산

```
Point #location2 = #Here ; 현재 위치를 위치 변수에 저장한다.
POINT   location1 = location2
        POINT pick = corner + pick
        POINT #place = #post
END
```

위치 변수를 수정하기 위해서는 trans 위치 변수인 Point loc_var1 = loc_var2 를 사용하거나, joint 위치 변수인 Point #pj1 = #pj2 를 사용해야합니다. 이러한 할당 작업의 경우, joint 위치 변수를 trans 위치 변수에 할당할 수는 없습니다. joint 위치 변수에서 trans 위치 변수를 얻기 위해서는 CvtTrans 라는 함수를 사용하여야 합니다.

- DECOMPOSE x[0] = part → 축 좌표가 6 축 일 때 x[0], x[1], x[2], x[3], x[4], x[5]
- DECOMPOSE angle[4] = #pick → angle[4] ~ angle[n]

DECOMPOSE 는 위치변수의 각각의 요소를 배열에 복사합니다. 각 위치 변수의 요소를 이용하기 위한 프로그램을 할 때 사용합니다.

```
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

현재 작업용 Tool, Base 좌표계를 설정합니다. Tool 과 Base 는 미리 UI 를 통해 만들어 놓은 다음 이를 선택하여 사용합니다. coreCon 메뉴 function 에서 다양한 방식으로 tool 과 base 를 설정할 수 있습니다.

2) Robot Motion

- MoveJ 위치명 → 약어 : MJ : Joint interpolation motion
- MoveL 위치명 → 약어 : ML : Straight line interpolation motion.
- MoveC 위치 1, 위치 2 → 약어 : MC : Circular interpolation motion.
- MoveX 위치명, 신호번호 → 약어 : MX

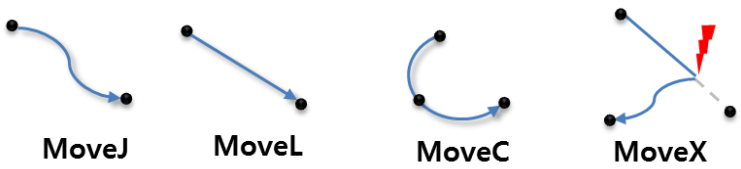


Figure 4 로봇모션

동작 명령어의 실행은 실행 후 동작이 시작되고 끝날 때까지 프로그램 실행이 멈추는 것이 아니라, 바로 다음 명령이 실행됩니다. 따라서, 다음과 같은 명령이 실행되면, 로봇이 #p2 로 움직이기 시작함과 동시에 DOut 신호가 출력되게 됩니다.

```
MoveJ #p2 ; 현재 위치에서 #p2 로 로봇이 출발
SetDO 2 ; 로봇이 출발하자마자 Digital Output 신호 2 번이 ON
MoveJ #p3 ; MoveJ #p2 의 동작이 끝나야 이 명령의 동작을 시작할 수
있으므로, MoveJ #p2 동작이 종료될 때까지 대기
```

- Delay 시간 : 로봇 동작 정지, MoveJ 처럼 Delay 동작중이라도 다음 스텝 명령은 동작 명령 전까지는 실행이 됩니다. 즉, 로봇 모션 명령어 이외의 명령어가 실행됩니다.
- Stable 시간

Delay 와 Stable 은 motion 명령어 입니다. Move 명령어 뒤에 작성된 Delay 와 Stable 명령어는 움직임이 끝날 때까지 기다립니다. Delay 는 단지 이전의 움직임이 끝난 후에 기다리는 작업을 수행하지만, Stable 은 주어진 시간동안 target 위치로 움직이라는 명령을

계속 실행합니다. 이 작업은 robot 움직임을 안정화 시키고 정확도를 증가시킵니다. motion 이 아닌 명령어는 Delay 와 Stable 명령어 뒤에 실행되는 non-motion 명령어는 Move 명령어와 같이 즉시 실행됩니다.

- ApproJ 위치명, 거리 : 거리는 Tool 의 Z 방향으로 특정거리만큼 이동
- ApproL 위치명, 거리 : Tool 의 Z 방향으로 특정거리만큼 직선 이동
- DepartJ 거리 / DepartL 거리 : Appro 명령과 반대로 Tool 의 z 방향으로 떨어지는 방향으로 이동하게 됩니다.

로봇은 아래 그림과 같이 Tool 의 z 방향이 Tool 이 진입하는 방향으로 잡게 됩니다. 특정 위치로 로봇을 빠르게 이동시킬 때 사용합니다. 따라서, 특정 거리만큼 현재 위치에서 Tool 을 진입시킬 때, Appro 명령을 사용하면 편리합니다.

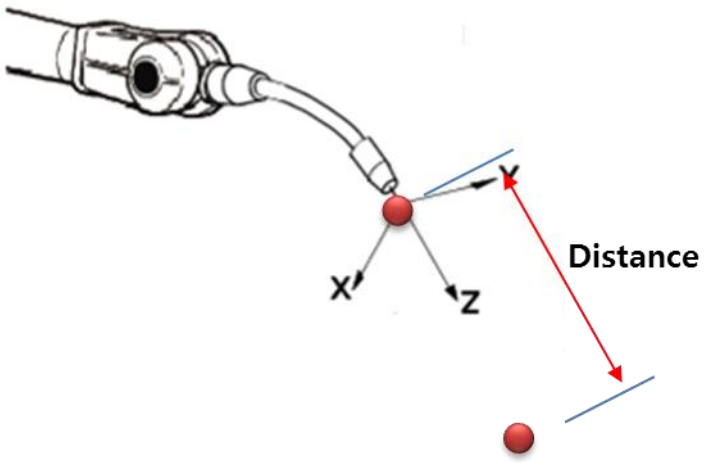


Figure 5 Appro 명령어 모션

- HOME / HOME2 : 2 개의 Home 위치를 지정할 수 있으며, 지정된 Home 위치로 이동하기 위한 명령입니다.

- ALIGN : TOOL 의 Z 방향을 가장 가까운 Base 의 좌표축과 맞춥니다.

- Move 명령 추가 인자 : MoveJ p1, [bundle signal no], [speed]
  - Bundle signal no : 로봇 동작 중에 signal 동작이 가능하도록 조건 및 signal set 를 bundle table 로 정의할 수 있다. 이렇게 정의된 bundle signal table 의 no 를 지정하여 사용하도록 하고, 매 동작 시 마다, speed 를 설정할 수 있다.
  - Speed 만 설정할 경우, Bundle signal no 는 0 또는 -1 을 입력한다.

- 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 : 주어진 축만 입력 값 만큼 이동
    - Drive axis_num, delta

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

3) Robot Setting

- Robot configuration : 6 축로봇의 자세를 지정하는 명령어입니다. 입력 위치가 Trans 위치일 경우, 여러 자세가 가능한데, 이 명령어를 이용하여 원하는 자세로 설정하는 것이 가능합니다. Scara 로봇의 경우, Lefty/Righty 가 있습니다. 그러나 직교나, 델타의 경우엔 사용되지 않습니다.
  - ABOVE
  - BELOW
  - LEFTY
  - RIGHTY
  - UWRIST
  - DWRIST
  - SINGLE
  - DOUBLE

- Motion Setting : 로봇의 움직임과 관련된 명령어입니다. 로봇의 속도와 Accuracy, Accel/Decel, Break 설정을 통해 로봇의 움직임을 제어할 수 있습니다.
  - Speed 속도 [Fixed] :
    - 속도를 설정합니다. Fixed 를 설정하면, 이 후 모든 속도가 지정된 속도로 되고, Fixed 를 사용하지 않으면, 다음 모션 명령의 속도만 변경됩니다.

```
MoveJ #p1   ; 최대 속도 100%로 동작
Speed 20    ;  다음 모션 명령의 속도는 최대속도의 20%
MoveJ #p2   ;  이 명령은 20%의 속도로 움직임
MoveJ #p3   ; 이 명령은 100%의 속도록 움직임

MoveJ #p1   ; 최대 속도 100%로 동작
Speed 20 Fixed   ;  이후  모션 명령의 속도는 최대속도의 20%
MoveJ #p2   ;  이 명령은 20%의 속도로 움직임
MoveJ #p3   ; 이 명령도  20%의 속도록 움직임

Speed 20 mm/s ; 20 mm/sec 로 속도가 설정됨
```

■ Accuracy 거리 [Fixed]

- : 로봇이 아래와 같이 P1→P2→P3 를 움직일 때, Accuracy 를 충분히 큰 값으로 정하게 되면, 로봇은 P2 점을 지나지 않고, 연속동작으로 부드럽게 지나게 된다. 경유점을 통과할 경우, 이러한 Accuracy 를 사용하게 되면, P2 에서의 가감속 구간을 제거할 수 있어 Cycle time 을 단축시킬 수 있다.

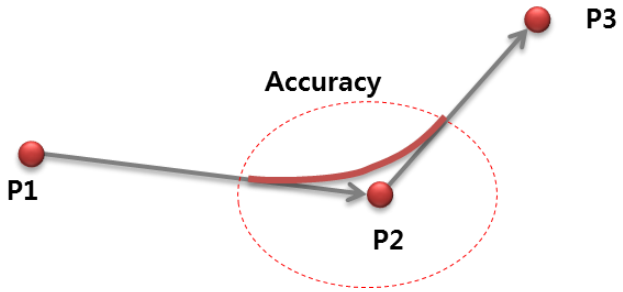


Figure 6 Accuracy 모션

■ Accel / Decel 가감속% [Fixed]

- : 로봇이 하중이 무거운 물체를 잡고 움직일 때, 기존의 동작으로 무리가 생길 우려가 있을 경우, 가감속 시간을 조정함으로써, 보다 유연한 모션을 구현할 수 있습니다. 최대 가속 시간은 로봇설정 파일에 정의되어 있습니다.
- 단위는 %이며, 최대가속도의 %가 입력됩니다. 입력 범위는 0.01 ~ 100% 가능합니다.

Accel 50 : 최대 가속도의 50% 사용, 속도가 천천히 증가되도록 하여, 진동 절감 효과를 가져올 수 있다. 다음 모션 명령만 변경됨

Accel 50 Fixed : 이후 모든 모션 명령의 가속 값이 변경됨

- Break : 로봇의 위치가 정확한 위치에 도달할 때까지 대기하고 있다가 도달한 후에 다음 스텝으로 진행하는 명령어입니다.

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

위와 같이 p2 와 p3 사이에 break 명령을 넣게 되면, p2 점에 도착할 때까지, MoveL p3 는 시작되지 않게 되어, 끊어진 모션이 수행되게 됩니다.

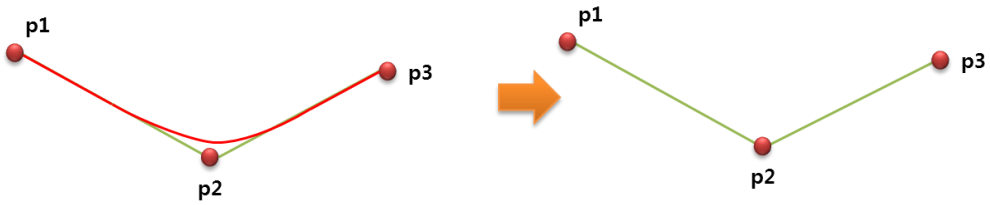


Figure 7 Break 삽입 시 모션 변화

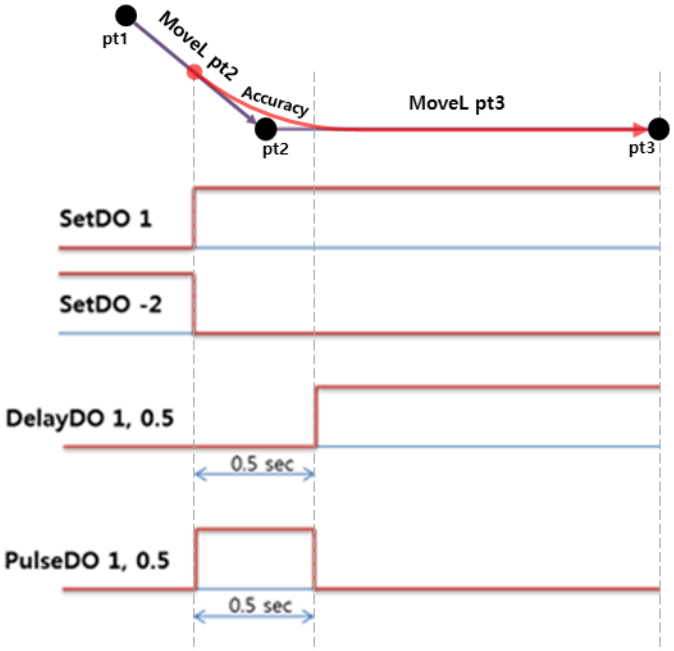
- Brake : 로봇을 정지시키고, 다음 스텝으로 이동하는 명령어입니다

4) I/O

- Reset : 모든 외부 출력을 OFF 하는 명령어입니다.
- SetDO digital_output_signal_#, ... : 출력 신호 설정하는 명령어입니다.
  - SetDO -sigval, 4 : sigval 신호 OFF, 4 번 신호 ON 합니다.
  - IO Number 는 출력 : 1~128, 입력 1001~1128 을 사용합니다.

- PulseDO signal#, time : 주어진 시간만큼 signal 을 ON 시킨 후, OFF 합니다.
- DelayDO signal#, time : 주어진 시간 후에 signal 을 출력합니다.
- RunMask startsignal#, 개수 : 실행 시만 signal 을 내보내고, 정지하면 바로 OFF 한다.
- Bits start, 개수 = value : 여러 개의 signal 을 설정하는 명령어입니다.
  - Bits 1,8 = 255 : 1~8 까지 신호가 연속 신호로 정의되고 출력 값은 255 가 된다.
- WaitSig signal#, ... : signal 조건이 만족할 때까지 wait

아래 그림은 MoveL pt2 명령을 수행한 후, SetDO, DelayDO, PulseDO 를 수행할 경우, 신호선의 변화를 나타낸 그림입니다. IO 동작 위치는 Accuracy 시작 부분입니다.



이러한 신호 처리 외에, 도착 5mm 전이나, 도착 0.5 초전, 혹은 출발 5mm 후 도착 5mm 전에 신호 출력과 같은 복잡한 설정으로 신호 타이밍을 조정하고자 할 때는 BsCondition, BsDo 를 통해, 시그널 번들 설정을 통해 적용할 수 있습니다.

5) Touch Probe 명령

- 위치 TouchEnable
  - TouchEnable TRUE 1 : 모든 probe ch 공통 type 의 touch 기능을 활성화..
  - TouchEnable TRUE 3, 4 : probe ch 별로 type 을 달리하여 기능을 활성화.
  - 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 축을 monitoring

- TouchStart probe1, probe2
  - Monitoring 시작, probe1, probe2
  - ex) TouchStart 1, 1 <- 1, 2 번 probe monitoring 시작

- TouchStop : monitoring 정지
- TouchWait timeout, result
  - 결과 wait : 1 쌍의 data 가 수집되면 result = true
- TouchDataCount(probe_index): Probe 에 감지된 count 를 return 합니다.

- 수집된 Touch 위치
  - ndata = TouchDataCount(probe_index)
  - #TouchJoint(Probe_index, Count_index)
  - TouchTrans(Probe_index, Count_index)

Example1 : Target point 변경

```
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 ; robot 을 target 으로 동작 시킨다.
TouchWait 2, result ; Enable 조건과, Touch Start 조건 성립 시, 다음 문장으로 전환
TouchStop ; Touch monitoring 종료
Brake ; Motion stop
; 획득한 데이터를 얻어 낸다.
npoint1 = TouchDataCount(1) ; touched point size
npoint2 = TouchDataCount(2) ; touched point size
IF result THEN
Point #ptouch1r = #TouchJoint(1, 1) ; RTOF 이므로 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
```

Example2 : Target point scanning → TouchWait 를 사용하지 않으면 된다.

```
TouchEnable TRUE, 3 ; Touch Enable , Rising edge 만 detect
TouchAxis 1, 0, 1, 0 ; Set axis to monitor : 1, 3 축만 i/o 선이 연결되었음
TouchStart 1, 0 ; probe 1 only start monitoring
MoveL ptarget ; robot 을 target 으로 동작 시킨다.
Break ; ptarget motion wait
;획득한 데이터를 얻어 낸다.
nloc = TouchDataCount(1) ; touched point size
for I = 1 to nloc ; rising, rising, ...
Point tp[i] = TouchTrans(1, i)
```

6) Network 명령

- Client Mode
  - TCPConnect socket 변수, IP address 문자열, port_number, timeout
  - TCPClose socket 변수
  - TCPRead socket 변수, 문자열 변수, [return code], timeout
  - TCPWrite socket 변수, 문자열데이터, [return code]

- Server Mode
  - TCPSStart soket 변수, port number
    - socket 변수가 음수가 반환되면, 에러가 발생한 경우입니다.
  - TCPSStop soket 변수
    - socket 이 close 됩니다.
  - TCPSAccept soket 변수, client_socket 변수
    - Client_socket 변수가 음수가 반환되면, 에러가 발생한 경우입니다.
  - TCPSCClose client_socket 변수
    - Client_socket 이 close 됩니다.
  - TCPSRead client_socket 변수, 문자변수, [에러코드변수], timeout
    - 연결된 client 로부터 읽기를 대기 합니다.
  - TCPSWrite client_socket 변수, 문자열, [에러코드변수]
    - 연결된 client 로 문자열을 전송합니다.

서버모드의 예제는 비전인터페이스 프로그램 예제를 살펴보기 바랍니다. 통신으로 전달 받은 문자열은 숫자 데이터로 변환할 필요가 있는데, 이 때 사용하는 함수가, 문자열을 토큰으로 분리하는 SplitStr 명령어와 , 문자열을 숫자로 바꾸어 주는 함수 Value 를 사용합니다.

<SplitStr 명령어 사용 예>

```
"%-2.1,15.4,95" 에서  tx = -2.1, ty = 15.4, trot = 95 를 얻어 내는 방법

 $msgread = "%-2.1,15.4,95"

SplitStr $token[0], $msgread, "%,"

     ;$msgread 에서 '%', ',' 두 글자를 분리 글자로 사용하게 합니다.

     ; $token[0] = "-2.1", $token[1] = "15.4", $token[2] = "95" 가 들어가게 됩니다.

Tx = Value($token[0])

Ty = Value($token[1])

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

7) Conveyor Tracking

- TkSetSig cvid, trigger signal #
  - Object queue 에 data 를 추가하기 위한 trigger signal 을 지정합니다.
  - Conveyor 설정에서도 정의할 수 있기 때문에, Program 에서 사용하지 않아도 됩니다.
  - Conveyor 설정에서 counter 의 latch 상태를 통해서, trigger signal 을 대신할 수도 있습니다.
- TkObjAdd cvid: 한번에 여러 개의 Object 를 인식하는 비전시스템인 경우, 처음 인식 시 하나의 Object 가 추가 되므로, 새로운 Object 을 추가할 경우 비전 데이터를 분석하여 추가할 수 있습니다.
- TkObjClear cvid: Object List 를 삭제합니다.
- TkObjDropCur cvid: Object 를 현 위치에서 Drop 합니다.
- TkObjGetType: Object Add 시 등록된 Type 을 얻어옵니다. 형상이 여러 개인 Object 를 인식하는 vision 시스템의 경우, 형상을 구별하기 위해, 변환 위치 뿐 아니라 type 도 입력해 두어야 하므로 입력된 type 에 따라, moving 위치를 구별하고자 할 때 사용됩니다.
- TkFilterNullShift: 프로그램 된 위치와 차이가 크지않은 데이터가 들어올 경우 Filtering(삭제)하는 기능입니다. 노이즈 성 Object 이 인식이 되면 비전 데이터는 합성이 안되기 때문에 중복 데이터가 발생할 수 있어 이를 삭제합니다.
- TkObjWait cvid, timeout, result : Conveyor 상의 object 가 작업 가능한 영역에 들어

올 때까지 대기 합니다. Timeout 값을 설정하면, 무한 대기가 아닌 주어진 시간만큼만 대기하게 됩니다. 이 때 Object 가 선택이 되면, result = TRUE 가 되고 object 선택에 실패하면, false 가 됩니다.

- TkMove cvid, pt : Conveyor 와의 동기 모션을 수행합니다. Linear move 로 이동하게 됩니다.
- TkAppro cvid, dist : 이전 모션의 Target point 혹은 현재 위치에서 tool z 방향으로 distance 만큼 진입합니다.
- TkDepart cvid, dist : 현재 위치에서 tool_z 방향으로 빠져 나오게 됩니다.
- TkMoveC cvid, p1, p2 : Conveyor 와 동기되는 MoveC 명령, 외부에서 보면, 타원을 그리게 됩니다.
- TkStop cvid : Conveyor 동기를 중지합니다.
- TkObjGet cvid, tx, ty, trot : 마지막에 삽입된 object 의 정보를 얻어 냅니다.
- TkObjShift cvid, tx, ty, [trot=0] [tz=0] : 마지막에 삽입된 object 의 보정위치를 설정한다. 비전으로 받은 데이터를 분석하여, 이 함수를 통해 이동량을 설정해 줍니다.
- TkObjRemove cvid : 마지막에 삽입된 object 를 삭제합니다.
- TkObjClone cvid, [result]
  - 마지막에 삽입된 objec 를 복사하여 추가합니다. 이 때, coveyor 정보만 복사되고, 보정량 정보는 0 으로 설정됩니다.
  - Result == TRUE 이면 복사가 성공했음, False 이면 복사에 실패했음을 의미합니다.
- TkObjFilterEnv cvid, enable_flag, check_radious, search_count
  - Object filtering 을 위한 설정을 합니다.
  - Check_radius 는 object 가 동일한 것인지 판단하는 거리 조건입니다.
  - Search count 는 동일 object 인지 판단하기 위한 검사 개수를 설정합니다.
- TkObjFilter cvid
  - 마지막에 삽입된 object 의 Filter 를 수행하여 동일한 object 이면 삭제합니다.
- Nobj = TkObjCount(cvid)
  - 작업이 가능한 object 의 개수를 return 합니다. 이 값으로 현재 작업해야 할 object 존재 여부를 판단할 수 있습니다.

<컨베이어 동기화 프로그램 예제 : 모션이 가능한 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)          ; object coordinate 점으로 좌표계의 원점
MoveL phome             ; 대기 위치 이동
TkObjWait convid        ; object 대기
; catch target - ppick is relative w.r.t the conveyor reference
TkMove convid,ppick
TkAppro convid,dist     ; ppick 에서 ToolZ 방향으로 dist 만큼 이동
Signal 1,2              ;
TkDepart convid,dist    ;
Break                   ; 현재는 Track 모션 종료시 반드시 삽입
MoveL pdrop             ; object drop
Signal -1,-2
```

- TkAppro/TkDepart 대신에 아래 처럼 해도 됨
  - Point ppick2 = Translate(ppick, 0, 0, 70)
  - TkMove cnvid, ppick2
  - TkMove cnvid, ppick
- 이 프로그램만 동작 시키면, Trigger 시그널이 작동 될 때 마다, 기준 Target 위치로 이동하게 됩니다. 변경이 필요할 경우엔 다음 예제인 비전 인터페이스를 적용합니다.
- **중간에 프로그램을 멈추고 다시 시작할 때는 Reset 을 수행하여 처음부터 프로그램을 다시 실행 하는 것이 안전합니다.**

비전 장치와 연결되어, 작업할 Object 의 위치를 변경해야 할 경우, SubTask 에서 아래와 같이 비전 정보를 이용하여, Object 의 위치를 변경시키는 프로그램을 동시에 실행하면 됩니다.

<비전인터페이스 프로그램 예제 : SubTask 에서 동시 실행>

```
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,"%, "    ;  형식 dx, dy 만 전달: "%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
```

● Trigger signal 에 의해 비전이 하나의 Object 에 대해, 평행 이동량만 인식할 경우의 예 입니다. 다수의 object 를 전송 시 Filter , Clone 등의 기능을 사용해야 합니다.

● 중간에 멈출 경우, TCP 연결 문제로 Reset 후 다시 실행하여야 합니다.

● 비전과의 통신만 테스할 경우, 아래의 TPWrite 명령으로 주고 받는 데이터를 확인할 수 있습니다.

■ TCPSRead cognex, $msgread, vis_ret

■ TPWrite 2, "Vision : %s", $msgread

● 고속으로 사용할 경우, 위와 같은 Diagnostic message 로 인한 시간 초과로 사용을 하지 않는 것이 좋습니다.

8) 기타

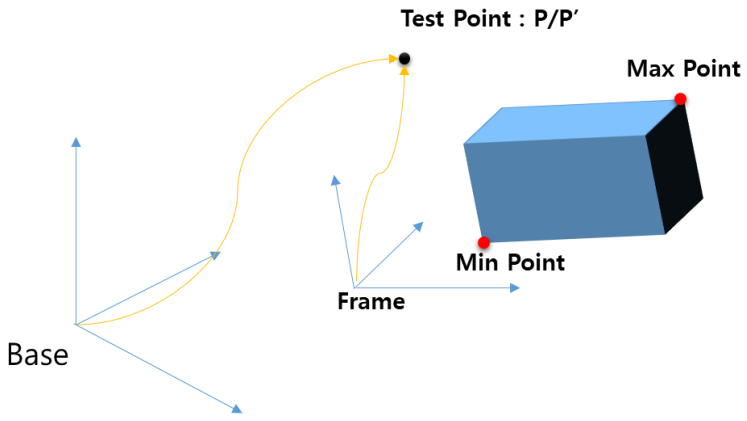
- ULIMIT / LLIMIT 각축 값 : upper limit, lower limit 설정
- TIMER

Timer 설정 : SetTimer timer#, time
Timer 읽기 : Timer(timer#)

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

9) World Zone

- Cartesian Limit Zone
- Joint 영역의 Limit 설정과는 별도로, Base 좌표계에 대하여 Box, Cylinder 의 조합으로 Limit 영역을 설정합니다.
- WZBox: Box 영역을 설정합니다.

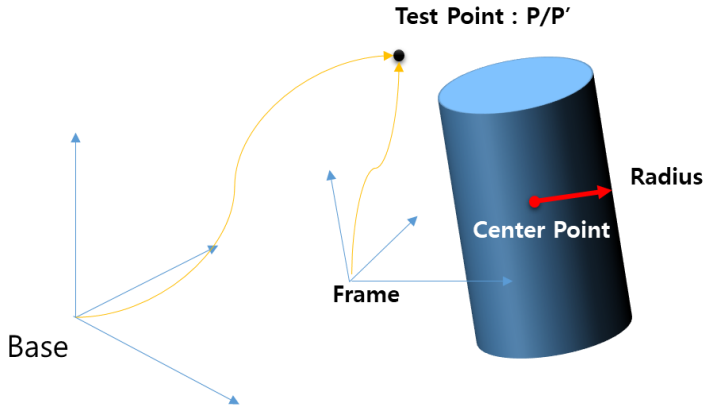


- WZBox id, operation, p1, p2, [frame]
  - Id: worldzone id
  - Operation:
    - 1: Add
    - -1 : Subtraction
  - P1: min point (x, y, z)
  - P2: max point(x, y, z)
  - Frame: WZone 좌표계(입력하지 않을 시 Base 와 동일하게 됨)

- WZBox example:

  WZBox 1, 1, Trans(10, 10, 0), Trans(1000, 500, 500), [Trans(Frame)]

- WZCheck On/Off: World Zone Check 를 On/Off 합니다.
- WZCyl: Cylinder 영역을 설정합니다.

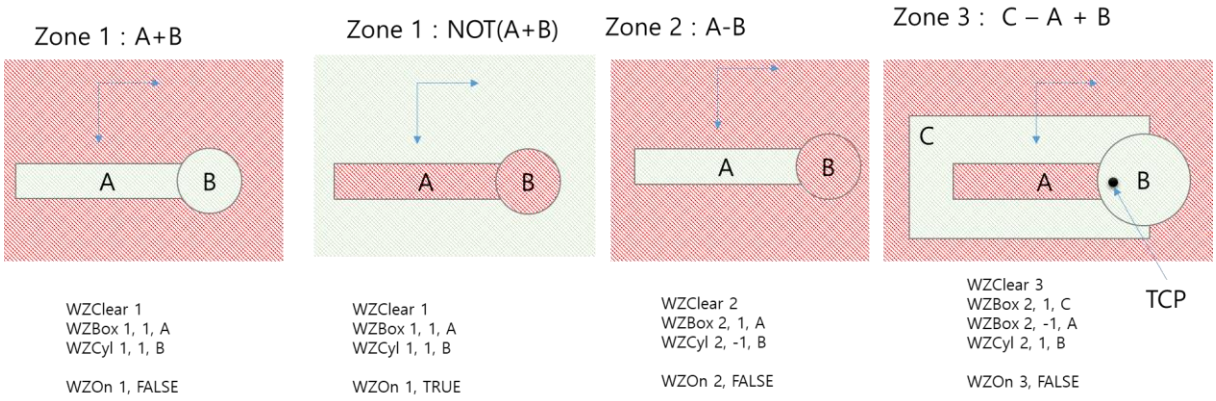


- WZCyl id, operation, cp, r, [frame]
  - Id: worldzone id
  - Operation:
    - 1: Add
    - -1 : Subtraction
  - cp: Center point(x, y, z)
  - r: radius
  - Frame: WZone 좌표계(입력하지 않을 시 Base 와 동일하게 됨)
- WZBox example:

  WZCyl 1, 1, Trans(500, 500, 0), 100, [Trans(Frame)]
  원통의 높이는 무한대입니다.

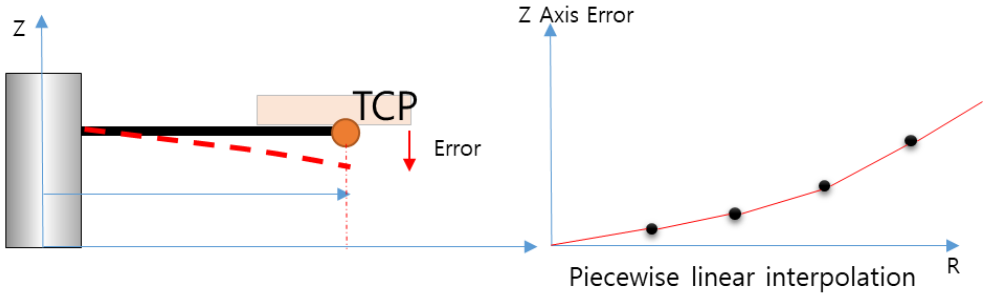
- WZClear id: 설정된 WZone 을 clear 합니다.
- WZOff id: WZone [id]를 비활성화 합니다.
- WZOn id, negate: WZone [id]를 활성화 합니다.
  - Negate: True/False
    - True 일 때 모든 영역이 동작 가능 영역이 되고 Operation 이 1(Add)이면 동작 불가 영역이 추가되고 Operation 이 -1(Subtraction)이면 동작 가능 영역이 추가됩니다.
    - False 일 때 모든 영역이 동작 불가능 영역이 되고 Operation 이 1(Add)이면 동작 가능 영역이 추가되고 Operation 이 -1(Subtraction)이면 동작 불가능 영역이 추가됩니다.
- WZSignal signal: WZ 에 TCP 가 들어올 경우 Signal 을 출력 할 수 있습니다.

- Composite Zone: Add, Subtraction



10) ZComp

- Z 축 방향의 에러를 Table 로 저장하고 programmed 모션에서 Z 축의 처짐을 보상합니다.



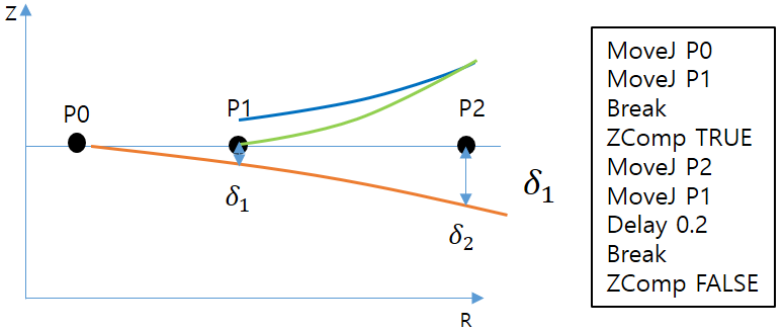
- TCP 위치에서 Z 축의 수직거리가 R 입니다.
- Z-axis 의 방향과 Error 의 방향은 반대가 되어야 합니다.
- Error Table

R	Error
500	0.5
600	0.65
800	0.8
1000	1

  - 위치: /mnt/mtd5/zcomp.ini
  - 파일 형식

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

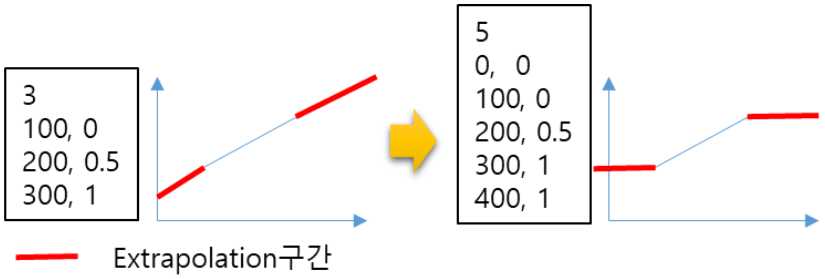
- ZComp TRUE/FALSE: ZComp TRUE/FALSE 으로 사용합니다.
- ZComp 사용시 주의사항
  - 1) Error 가 존재하는 구간(P1~P2)에서 Enable/Disable 시킬 경우 왕복 궤적이 달라질 수 있습니다.



위 그림에서 처럼 P1 에서 P2 로 이동시 P1 에서 δ_1의 Error 가 존재하기 때문에 동작 계획은 (P1- δ_1) -> P2 가 됩니다. 이때 P2 로 가기 위해서 실제로 (P2+ δ_2)의 위치로 이동합니다.

P2 에서 P1 으로 올 때는 (P1+ δ_1)의 위치로 이동합니다. 따라서, Error 가 존재하는 구간에서 Enable/Disable 시키면 왕복 궤적이 달라질 수 있습니다.

  - 2) 동작 중 Enable/Disable 시킬 경우
    ZComp 상태(Error 보정상태/Error 무보정 상태)가 변경될 수 있기 때문에 상태 전이 시에는 , Break 등 명령을 사용하여 동작을 완료한 수에 변경하여야 합니다.
  - 3) Extrapolation
    Error Table 설정 이외의 구간에서 Extrapolation 을 하기 때문에 Error 정의 구간 밖을 상수로 두고자 할 경우우, Error 의 변경이 없는 점을 추가해 주어야 합니다.



11) User Coordinate 관련 명령어

- UCMove #ptarget : User coordinate 상에서 정의된 Linear motion 을 수행합니다.
- UCMasterArm arm1, [arm2], [arm3], [arm4] : User coordinate 상에서 모션이 해결되어야 하는 우선순위를 정의합니다.
  - 2 개의 user coordinate set 가 구성된다면 UCMasterArm1 혹은 UCMasterArm 2 를 통해 우선적으로 모션이 유지되어야 할 ARM 을 정의하고, 나머지 ARM 은 잔여

여유도를 통해 위치를 결정하도록 합니다.

- UCHint $hint_name, value : Custom ARM 에 필요한 옵션을 설정합니다. Custom 기구학 모듈에 전달 되기 때문에, 이를 hint 로 위치나 motion 결정을 할 수 있습니다.

9. Diagnostic Functions

- TPWrite icon, str_format, [arg1], [arg2], [arg3], ..., [arg10] : message out 영역에 출력합니다. Printf 와 같은 형식을 사용하며, ument 개수는 10 개까지 가능합니다.
  - icon = 0 : error 1 : warning, 2 : information 이 있으며, Error 에 해당하는 message 는 system log 에도 기록이 되게 됩니다.
  - ICONE = 0, ICONW = 1, ICONI = 2 로 정의되어 있어 TPWrite ICONI, "message" 로 사용해도 됩니다.
  - cycle = cycle + 1
  - TPWrite 2, "program cycle = %d", cycle
- TPClear : message out 영역을 clear 합니다.
- RaiseError user_erro_code : User error code 에 해당하는 error 가 설정되며, robot 은 정지되게 되고, Log 에 기록이 남게 됩니다.
  - User error code 는 -9001 ~ -10000 까지 사용할 수 있습니다.
  - ex) RaiseError -9001
- Error 정지 시 Auto Restart 하는 법
  - SubTask 에서 Error output signal 을 Detect 합니다.
  - SubTask 에서 Error reset 을 보냅니다.
- AbortOnUserErr
  - Subtask 에서 User Error 발생시 Subtask 정지 여부를 결정합니다.
  - Subtask 에서 User Error 발생시 Subtask 비정지, Maintask 정지합니다.
  - AbortOnUserErr On 으로 설정하면 Subtask 에서 User Error 발생시 Subtask 도 정지합니다.
- AutoSleep: On/Off 로 Loop 명령어에서 1 cycle sleep 을 조정합니다.
  - 반복문 사용시 주의할 점은 While 이나 GOTO 문 등 끝나는 시점이 불투명한 상황에서는 WaitTime(혹은 시간 지연을 줄 수 있는 명령)을 삽입해야 합니다(최소 1 cycle time, 2 msec), 사용상 주의가 필요합니다.

10. 외부 IO를 통한 프로그램 선택

PLC 등의 외부 제어기에서 로봇 프로그램을 선택하고 실행할 수 있도록 외부 IO 를 통한 프로그램을 설정은 전용 신호 설정과, 다음의 두개의 명령어를 통해 이루어 집니다.

Macro 명령어

EPSmode On or EPSmode Off

external program selection 모드의 사용 여부를 결정합니다.

EPSWait

외부에서 program 을 선택할 수 있도록 대기 합니다.

EPS 를 사용하기 위한 전용신호는 적어도 6 종류의 IO 를 설정해 주어야 합니다.

Robot 용 .conf 에서 아래와 같이 설정한다.

Config 설정 예제)

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

- Example 은 1009~ 1012 까지 IO 접점을 사용하도록 설정

EPS_START_BIT ~ EPS_END_BIT 은 프로그램 지정을 위한 IO bit 수를 정의합니다. 예를 들어, 1009~ 1012 로 설정하면 4bit 크기의 프로그램을 설정할 수 있게 됩니다.

1 ~ 9 까지의 프로그램은 Pg1 ~ Pg9 로 명칭이 정의된 프로그램이 선택되고,

10 ~ 99 까지의 프로그램은 Pg10 ~ Pg99

100 ~ 999 까지의 프로그램은 Pg100 ~ Pg999 가 선택되게 됩니다.

EPSMode 가 ON 상태이고, EPSWait 문장이 실행 되어 대기하게 되면, EPS_STATUS 가 ON 이 됩니다, 즉, 외부 PLC 에서는 EPS_STATUS 를 확인 하여, ON 일 경우, 프로그램을 선택합니다. 이 때, EPS_ON 을 ON 시키면, 설정된 프로그램이 실행되고, EPS_OFF 를 ON 시키면, EPSWait 가 취소가 되어, 로봇 프로그램이 다음 스텝이 실행되게 합니다.

1) EPS MODE를 사용한 프로그램 호출 예제

- Macro 예제 EPS Macro (EPS 설정 Macro Program name: EPSGO)

```
HOME
EPSMode ON
EPSWait
```

- Macro 예제 pg1 (Macro Program name : Pg1)

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

- Macro 예제 pg2 (Macro Program name: Pg2)

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

- 예제에 대한 설명
  - .EPSGO 를 실행 하면 EPSMode 가 ON 이 된다.
  - EPSWait 에서 EPS_ON 입력을 기다린다.
  - Macro 선택 입력을
    1012 = off, 1011 = off, 1010 = off, 1009 = on 와 같이 입력 한다
  - EPS_ON 을 입력 시키면 Pg1 이 실행된다.
  - Macro 선택 입력을
    1012 = off, 1011 = off, 1010 = on, 1009 = off 와 같이 입력 한다.
  - EPS_ON Input 을 누르면 pg2 가 실행된다.
  - EPS_ON Input 이 없을 시 무한 대기한다.

2) Bits와 SWITCH CASE문을 사용한 프로그램 호출 예제

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

3) EPSWait 조건문을 사용한 프로그램 호출 예제

- Macro 예제 EPS Macro (EPS 설정 Macro Program name: EPSGO)

```
Result = 0
EPSMode ON          ;EPSMode 를 ON 합니다.
EPSWait 2,result    ;2 초간 EPS 신호가 없을경우 result = true 를 반환
IF result == TRUE THEN
Call other()        ;other()프로그램을 실행
END
```

- Macro 예제 pg1 (Macro Program name : Pg1)

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

- Macro 예제 pg2 (Macro Program name: Pg2)

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

- Macro 예제 other (Macro Program name: other)

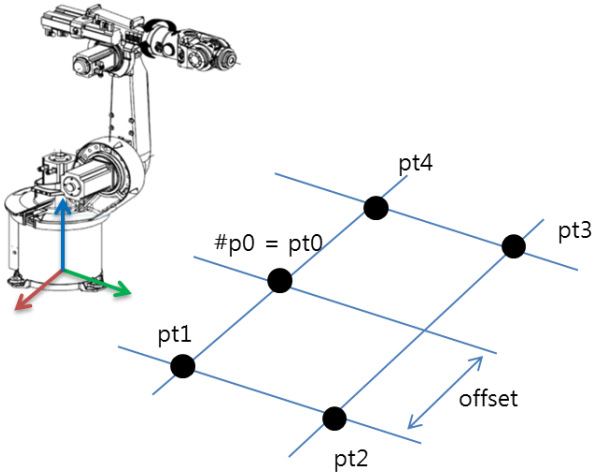
```
MoveJ #p5
MoveJ #p6
```

- 예제에 대한 설명
  - .EPSGO 를 실행 하면 EPSMode 가 ON 이 된다.
  - EPSWait 에서 EPS_ON 입력을 기다린다. .
  - Macro 선택 입력을
    1012 = off, 1011 = off, 1010 = off, 1009 = on 와 같이 입력 한다
  - 2 초 안에 EPS_ON 을 입력 시키면 Pg1 이 실행된다.
  - Macro 선택 입력을
    1012 = off, 1011 = off, 1010 = on, 1009 = off 와 같이 입력 한다.
  - 2 초 안에 EPS_ON Input 을 누르면 pg2 가 실행된다..
  - 2 초동안 EPS_ON Input 을 누르지 않을 경우 other()프로그램이 실행된다

11. 예제 프로그램

1) 기본 모션 프로그램

#p0 를 교시점으로
Offset = 변수 설정
#p0 → pt1 → pt2 → pt3 → pt4 → pt0 로 이동하는 프로그램 예 입니다.




```
MoveJ #p0     ;  적절한 위치를 교시하여 저장합니다.

Offset = 100     ; offset 량 지정

Point pt0 = CvtTrans(#p0)        ;  Joint 위치변수를 Trans 위치 변수로 변경

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
```

2) CP Motion

Pt2 점의 Accuracy 를 증가 시켜, 부드러운 모션이 되게 하고, 로드 적재를 예상하여, 가감속을 줄여 줍니다.

```
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       ; pt2 의 동작에만 영향을 미칩니다.

MoveL pt3       ; pt3 의 동작엔 원래의 설정으로 복원됩니다.

MoveL pt4

MoveL pt0
```

- 전체 설정을 변경하고자 할 때는 Accuracy 50 Fixed, Accel 50 Fixed, Decel 50 Fixed 의 형태로 사용합니다. 설정값이 고정되어 이 후 스텝에 계속 영향을 미치게 됩니다.

3) IO 사용

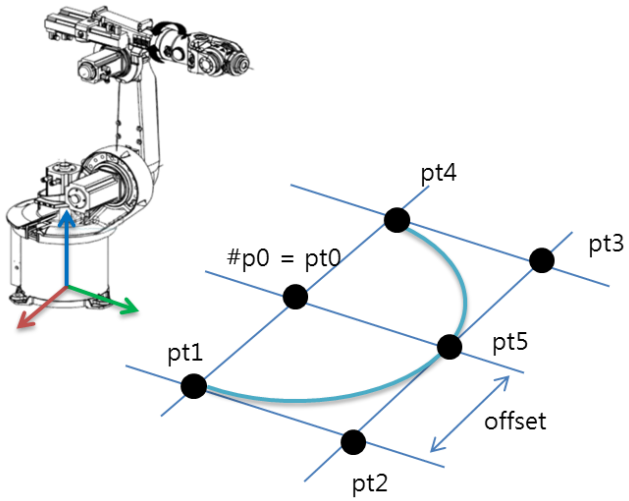
Pt1 출발 시에, Digital Input 1002 번의 신호가 ON 이여야 하고, Pt2 점과 Pt3 점 사이를 이동할 때, Digital output 3, 5 의 신호가 각각 ON 이 되게 합니다.

```
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     ; Digital Input 1002 를 대기
MoveL pt1
Accuracy 50
Accel 50
Decel 50
MoveL pt2
SetDO 3, 5       ; pt3 로 출발하면서, 출력 신호를 On 시킵니다.
MoveL pt3
SetDO -3, -5     ; 출력 신호를 OFF 시킵니다.
MoveL pt4
MoveL pt0
```

- DelayDO, PulseDO 등을 이용하여, 지연 출력 및, Pulse 출력을 할 수 있습니다.
- Bundle IO 조건을 정의하여, 동작의 일부 구간에만 IO 를 출력할 수 있습니다.

4) MoveC

직선 이동 대신에, pt1 에서 pt5 를 거쳐 pt4 로 이동하는 원호보간을 하는 프로그램입니다.



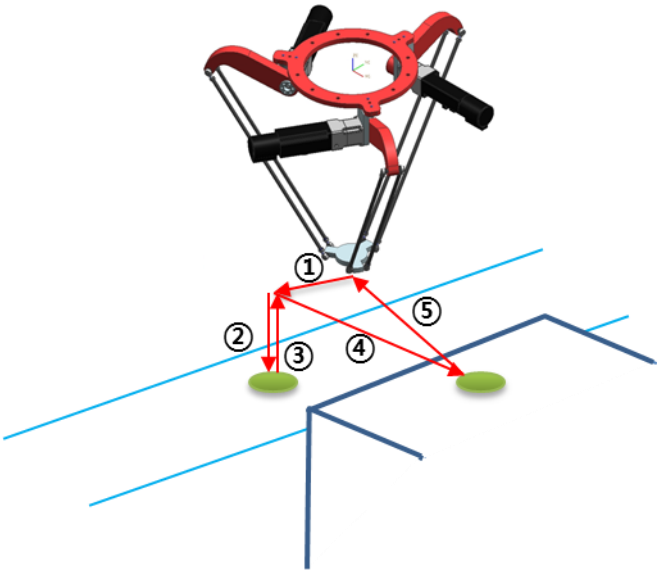

```
MoveJ #p0

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 ; 원호보간은 경유점과, 종착점을 입력하여야 합니다.

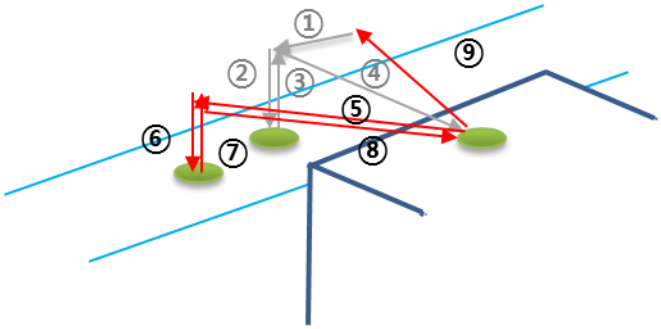
```
MoveL pt0
```

5) Conveyor Tracking and Pickup

작업 순서가 아래 그림과 같이 되고, 컨베이어에 다수의 작업물이 대기 중일 때, 로봇의 대기 위치로 이동하지 않고, 연속적인 픽업 작업을 수행하게 하는 프로그램입니다.



작업 대기물이 없을 때, 1→2→3→4→5 의 순서로 작업



작업 대기물이 있을 때, 5 의 순서가 대기위치로 가지 않고, 직접, 다음 작업으로 이동하게 함

```
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   ; 대기물이 없으므로, 대기 위치로 이동
        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   ;   대기물이 있으므로 home 을 가지 않고 연속작업으로
            GOTO 200
        END
```

Track 모션 시작전에는 반드시, Break 를 입력하여, 두 모드를 끊어 주어야 합니다.

12. Function Reference

Abs – 절대 값을 할당합니다.

Description:

인자 값을 절대 값으로 변환하여 할당합니다.

Syntax:

value1 = Abs(value2)

Return value: value1

Data type : Number

양수 음수 관계 없이 무조건 양수로 데이터를 받습니다.

Arguments: value2

Data type : Number

양수 음수와 관계 없이 입력할 수 있으며 number 변수가 아니어도 직접 숫자 데이터를 입력할 수 있습니다.

Example:

a1= -30

a2 = 10

dist = Abs(a1 – a2)

dist 에 'a1 – a2' 된 절대 값으로 할당됩니다.

Result:

dist : 40

ACos – arc cosine 값을 계산하여 할당합니다.

Description:

입력한 수치의 arc cosine 값을 계산하여 할당합니다.

Syntax:

angle = ACos(value)

Return value: angle

Data type : Number

Arguments: value

Data type : Number

Example1:

acosdg1= ACos(0)

acosdg1 에 arc cosine 90 의 값을 할당합니다.

AIn – Analog 입력을 할당합니다.

Description:

설정한 channel 에 입력된 analog 값을 할당합니다..

Syntax:

value1 = AIn(channel)

Return value: value1

Data type : Number

Arguments: channel

Data type : Number

AI 의 channel number 를 입력합니다.

Example:

aichannel2 = AIn(2)

aichannel2 에 2 번 채널의 analog Input 값이 할당됩니다.

AOut – Analog 출력을 할당합니다.

Description:

설정한 channel 에 출력된 analog 값을 할당합니다..

Syntax:

value1 = AOut(channel)

Return value: value1

Data type : Number

Arguments: channel

Data type : Number

AO 의 channel number 를 입력합니다.

Example:

aochannel2= AOut(2)

aochannel2 에 2 번 채널의 analog output 값이 할당됩니다.

Asc – ASCII code 값을 할당합니다.

Description:

String index 에 해당하는 ascii 값을 할당합니다.

Syntax:

value1 = Asc("CoreRobot", index)

Return value: value1

Data type : Number

Arguments: "CoreRobot", index

Data type : $String, Number

$String 변수를 입력할 수 있습니다, ascii 로 받고 싶은 위치의 index 를 입력합니다.

Example:

ascii1 = Asc("CoreRobot", 1)

ascii1 에 첫 번째 index 인 'C'의 ascii 값을 할당합니다.

Result:

ascii1 : 67

ASin – arc sine 값을 계산하여 할당합니다.

Description:

입력한 수치의 arc sine 값을 계산하여 할당합니다.

Syntax:

angle = ASin(value)

Return value: angle

Data type : Number

Arguments: value

Data type : Number, 각도 단위는 degree 입니다.

Example1:

asinedg1= ASin(1)

asinedg1 에 arc sine 90 의 값을 할당합니다.

Atan2 – Arc tangent2 값을 계산합니다.

Description:

Arc tangent Y/X 값을 할당합니다.

Syntax:

value1 = Atan2(y, x)

Return value: value1

Data type : Number

Arguments: y, x

Data type : Number

Y 축과 X 축 좌표를 입력합니다.

Example:

y = 1

x = 1

degree1 = Atan2(y,x)

degree1 에 Arc tangent y/x 값을 할당합니다.

Result:

degree1 : 45

Bits – bits 값을 숫자로 변환합니다.

Description:

연속된 개수의 디지털 신호를 읽어 숫자로 변환하여 할당합니다..

Syntax:

value1 = Bits(signal, amount)

Return value: value1

Data type : Number

Arguments: signal, amount

Data type : Number

Example1:

cnum = Bits(1009, 4)

Bits 로 부터 변환되어 받은 값을 cnum 에 할당합니다.

Example2:

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

DI 1012, 1011, 1010, 1009 이 할당 됩니다

1012 = off, 1011= Off, 1010 = On, 1009 = off 되었을 때 bit 로 0010 이 됩니다

0010 을 10 진수로 변환하여 progrum 에 num 2 이 할당되어 CASE 2 가 실행됩니다.

CheckSum – Serial 통신에서 통신 데이터의 유효성을 검사합니다.

Description:

Serial 통신에서 데이터를 전송할 때 전송된 데이터에 오류가 있는지를 확인하여야 합니다. 데이터 전송 오류를 체크 하는 방법으로 CRC16, VRC, LRC(BCC), SUM, OR 총 5 가지 방법을 제공하고 있습니다..

Syntax:

chkvalue = CheckSum($data, type)

Return value: chkvalue

Data type : Number(Dec)

Arguments: $data, type

Data type : String, Number

type:

0 : CRC16

1 : VRC(수평 XOR)

2 : LRC(BCC)(수직 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

그러므로 chkvalue = 49+50+51 = 150 입니다.

$Chr – Hex code를 Ascii Char로 변환하여 String으로 반환합니다.

Description:

Hex code 를 Ascii Char 로 변환하여 String 으로 반환합니다.

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"

Cos – Cosine 값을 계산합니다.

Description:

입력한 수치의 Cosine 값을 계산하여 할당합니다.

Syntax:

value1 = Cos(value2)

Return value: value1

Data type : number

Arguments: value2

Data type : Number

Example1:

cosdg1= Cos(90)

cosdg1 에 Cosine 90 의 값을 할당합니다.

Result:

cosdg1 : 0

CvtTrans – Joint Point를 Trans Point로 변환합니다.

Description:

사용하고 있는 base 와 tool 좌표 계 기준으로
Joint Point 값을 Trans Point 값으로 변환하여 할당합니다..

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)
#joint1 의 좌표 값을 Trans 좌표로 변환하여 ctrans1 에 할당합니다.

Dest/#Dest – 현재 도착 위치 좌표를 할당합니다.

Description:

현재 계획된 모션의 도착점 위치를 Trans 혹은 Joint 형식의 변수에 할당합니다.

Syntax:

Point #value1 = #Dest

Point value2 = Dest

Return value: value1, #value2

Data type : #Joint or Trans

Trans 는 Dest 를 사용하고 Joint 는 #Dest 를 사용하여 값을 받습니다.

Example1:

Point dtrans1 = dest

Point dJoint1 = #dest

dtrans1 에 현재 위치 값을 Trans 변수로 할당합니다

#djoint1 에 현재 위치 값을 Joint 변수로 할당합니다

Distance – A와 B사이의 거리 값을 구합니다.

Description:

A 와 B 사이의 거리를 구하여 할당합니다.

Syntax:

value1 = Distance(A, B)

Return value: value1

Data type : Numver

Arguments: A, B

Data type : Trans, Trans

시작 위치와 끝 위치를 입력합니다.

Example1:

Point dtran1 = Trans(10)

Point dtans2 = Trans(20)

distrance1 = Distance(dtran1, dtran2)

dtran1 과 dtran2 사이의 거리를 구하여 distrance1 에 할당합니다.

Result:

distrance1 : 10

DX, DY, DZ – X or Y or Z in Trans 변수 좌표를 할당합니다.

Description:

Trans 변수에서 X or Y or Z 좌표를 얻어 Number 에 할당합니다.

Syntax:

value1 = DX(transvar)

value2 = DY(transvar)

value3 = DZ(transvar)

Return value: value1, 2, 3

Data type : Numver

Arguments: transvar

Data type : Trans

DX 는 Delta X 좌표, DY 는 Delta Y 좌표, DZ 는 Delta Z 좌표를 추출합니다.

Example1:

Point dtran1 = Trans(10, 20, 30)

dxt1 = DX(dtran1)

dyt1 = DY(dtran1)

dzt1 = DZ(dtran1)

dxt1 에 dtran1 의 X 좌표가 할당된다

dyt1 에 dtran1 의 Y 좌표가 할당된다

dzt1 에 dtran1 의 Z 좌표가 할당된다

Result:

dxt1 : 10, dyt1 : 20, dzt1 : 30

ErrorCode – User Error code를 반환합니다.

Description:

User Error code 를 반환합니다..

Syntax:

value1 = ErrorCode

범위는 (-9001 ~ -10000)까지 사용가능합니다.

Return value: value1

Data type : Number

Example1:

a = ErrorCode

a 에 UserError code 가 반환됩니다.

Frame – frame 좌표계를 구하여 할당합니다.

Description:

원점과 X 좌표 ,XY 의 평면, Z 축 방향을 입력, 좌표계 frame 을 구하여 할당합니다,

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

첫 번째 인자 값은 원점, 두 번째 인자 값은 X 방향,

세 번째 인자 값은 Y 축이 존재하는 평면을 정의,

네 번째 인자 값은 Z 축 방향을 입력합니다.

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)

newcoord 에 prog, px, pxy, pz 기준의 frame 좌표가 할당됩니다.

Result:

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

$Fill– 문자열에 버퍼를 할당합니다.

Description:

UFunc 기능을 사용하여 문자를 In/Out 할 때 문자열의 버퍼 크기를 할당하기 위해 사용합니다.

Syntax:

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

Return value: $string

Data type : String

Arguments: "value", size

Data type : $String, Number
value 를 입력하고 size 만큼 버퍼를 할당합니다.
Max Size 253

Example1:

```
num_size = 200
$str_var = $Fill("a", num_size)
Ufunc 1, 0,0,0,$str_var
```

num_size 에 200 입력, $str_var 에 num_size 만큼 버퍼 할당.
variable 를 확인하면 문자 a 가 200 개 할당되어있다.
UFunc 에서 $str_var 의 버퍼 크기 만큼 문자열을 리턴 받는다.

$HexStr – Ascii Char를 hex code로 변환하여 String으로 반환합니다.

Description:

Char 를 hex code 로 변환하여 String 으로 반환합니다.

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

Char 값(abcd1234)을 $str 에 할당합니다.

$str 을 Hex_code 로 변환하여 $hex_code 에 할당합니다.

$hec_code: 313233

Here/#Here – 현재 위치를 할당합니다.

Description:

현재의 Joint 혹은 Trans 위치를 할당합니다.

Syntax:

Point #value1 = #Here

Point value2 = Here

Return value: #value1 or value2

Data type : #Joint or Trans

Here 는 Trans, #Here 는 Joint 를 할당합니다.

Example1:

Point gotrans1 = Trans(10,10,10)

MoveL gotrans1

Point tphere1 = Here

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

MoveJ #gojoint1

Point #jphere1 = #Here

tphere1 에 gotrans1 으로 이동한 위치 좌표가 할당됩니다

#jphere1 에 gojoint1 으로 이동한 위치 좌표가 할당됩니다.

Result:

tphere1 : 10,10,10

#jphere1 : 20,20,20

#Joint – Joint 변수를 생성합니다.

Description:

Joint 변수를 생성합니다.

Syntax:

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

Return value: #value1

Data type : #Joint

Arguments: j1 ~ 6

Data type : Number

J1~6 은 Joint1~ 6 번째와 같습니다.

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)

#joint1 이 n1,n2,n3,n4,n5,n6 의 값이 할당된 Joint 변수로 생성된다.

Result:

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

JVal – Joint 변수에 입력된 특정 Joint 값을 할당합니다.

Description:

Joint 변수의 특정 Joint 값을 return 받는다.

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

Joint 변수 값과, 원하는 joint index 를 입력하여 값을 return 받을 수 있습니다.

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)

val_join1 에 10 이 return 됩니다

val_join2 에 30 이 return 됩니다

val_join3 에 50 이 return 됩니다

$Int16B – hex code를 2자리 Ascii char로 변환하여 할당합니다.

Description:

Hex code 를 2 자리 ascii char 로 변환하여 할당합니다..

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

Len – String의 길이를 할당합니다.

Description:

문자열의 길이를 구하여 할당합니다.

Syntax:

value1 = Len("CoreRobot")

Return value: value1

Data type : Number

Arguments: "CoreRobot"

Data type : $String

$String 변수를 입력할 수 있습니다.

Example1:

length1 = Len("CoreRobot")

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

length1 에 "CoreRobot" 문자열 길이가 할당됩니다

length2 에 $corerobot 변수 내부에 할당되어있는 문자열의 길이가 할당됩니다.

Result:

length1 : 9 length2 : 15

$Mid – String을 부분적으로 추출하여 할당합니다..

Description:

문자열에 지정된 index 에서 시작하여 n 개의 문자가 포함된 문자열을 할당합니다..

Syntax:

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

Return value: $value1

Data type : $String

Arguments: "CoreRobot", index1, count1

Data type : $String, Number, Number

"CoreRobot"에 $String 변수를 입력할 수 있습니다, index1 은 받을 문자열의 위치를 입력하고 count1 은 받을 만큼의 문자열 개수를 입력합니다.

Example1:

count1 = 3

index1 = 2

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

$value1 에 index1 에 지정된 위치부터 count1 만큼의 String 이 할당됩니다.

Result:

$value1 : ore

Random = 임의의 값을 할당합니다.

Description:

지정한 Number 에 범위 0~1 의 임의의 값을 할당합니다.

Syntax:

value1 = Random

Return value: value1

Data type : Number

Example1:

random1 = Random

random1 에 임의의 값을 할당합니다.

Round – 반올림 값을 계산하여 할당합니다.

Description:

반올림 값을 계산하여 할당합니다..

Syntax:

value1 = Round(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number

Example1:

n1 = 3.6

round1 = Round(n1)

round1 에 n1 이 반올림 된 값을 할당합니다.

Result:

round1 : 4

Rx, Ry, Rz – 지정한 각도만큼 회전한 Trans 값을 할당합니다.

Description:

Number 에 지정한 각도만큼 회전한 회전 Trans matrix 값을 할당합니다..

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, 각도 단위는 degree 입니다.

Example1:

angle = 90

Point tf = Rx(angle)

Rx 를 정의하는 변환 행렬은

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(angle) & -\sin(angle) \\ 0 & \sin(angle) & \cos(angle) \end{bmatrix}$$

이 되기 때문에 Rx(90)의 경우

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

이 됩니다.

Sig – 인자에 입력된 신호의 결과를 할당합니다.

Description:

인자에 입/출력 된 신호의 AND 결과를 할당합니다.

인자에 입/출력 된 신호의 상태 값을 할당합니다.

Syntax:

value1 = Sig(1001, 1002)

statussig = Sig(1007)

Return value: value1, statussig

Data type : Number

Number 는 -1 또는 0 을 할당합니다

-1 은 Ture(on), 0 은 False(off)입니다.

Arguments: 1001, 1002, 1007

Data type : Digital Input Number

1001~1128 : Input

Example1:

sig1 = Sig(1001, 1002)

statussig = Sig(1007)

sig1 에 Input 1001 과 1002 가 AND 된 결과 값을 할당합니다.

statussig 에 Input 1007 의 상태 값을 할당합니다

Result:

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

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

Sin – Sine 값을 계산하여 할당합니다.

Description:

입력한 수치의 Sine 값을 계산하여 할당합니다.

Syntax:

value1 = Sin(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number, 각도 단위는 degree 입니다.

Example1:

sinedg1= Sin(90)

sinedg1 에 Sine 90 의 값을 할당합니다.

Result:

sinedg1 : 1

Sqrt – 제곱근을 계산하여 할당합니다.

Description:

입력한 Number 의 제곱근을 구하여 Number 에 할당합니다.

Syntax:

value1 = Sqrt(value2)

Return value: value1

Data type : Number

Arguments: value2

Data type : Number

Example1:

n1 = 4

sqrt1 = Sqrt(n1)

sqrt1 에 4 의 제곱근을 할당합니다.

Result:

sqrt1 = 2

Timer – 지정 Timer의 값을 읽어 옵니다.

Description:

설정된 id 에 해당하는 Time 값을 할당합니다.

SetTimer 와 함께 사용하며, Id 는 1~9 까지 사용할 수 있습니다. SetTimer 로 시간을 설정하고, 흘러간 시간을 Timer 를 이용하여 측정합니다.

Syntax:

elapsed_time = Timer(timer_id)

Return value:

Data type : Number : 현재 시간 값을 초단위로 반환합니다.

Arguments:

timer_id : Data type : Number 타이머 id

Example1:

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

1 번 타이머를 사용하여, 프로그램의 사이클 타임을 측정하는 예제입니다. SetTimer 로 Timer 를 0 으로 설정한 후, Timer 함수를 통해 흘러간 시간을 측정하여, timer1 변수에 할당합니다.

#TouchJoint/TouchTrans – TouchProbe가 감지된 위치를 할당합니다.

Description:

Motor driver 에 있는 Touch signal 용 IO 에 touch(signal)가 감지 되었을 때 현재 motor 의 위치 값을 Joint 혹은 Trans 에 할당합니다. WaitSig 명령과 Here 명령을 사용하여 위치를 얻어 내는 것 보다, 로봇이 동작중에도 정확한 위치를 얻을 수 있습니다.

Syntax:

Point #p_value = #TouchJoint(Probe_index, Count_index)

: Joint 위치를 얻어 냅니다.

Point t_value2 = TouchTrans(Probe_index, Count_index)

: Trans 위치를 얻어 냅니다.

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 번 axis Monitoring

TouchStart 1, 1 → probe 1, 2 channel 사용

MoveJ #position1 → 동작 중 Touchprobe 에 Rising, Falling 감지

TouchWait 1, result → Rising 1, Falling 1 번 쌍으로 감지 시 result 가 TRUE

TouchStop → Touch Scanning stop

IF result THEN

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

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

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

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

END

Trans – Trans 변수를 생성합니다.

Description:

Trans 변수를 생성합니다.

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

Trans 좌표 값을 입력. 자세를 표현하는 euler angle 값은 0, 0, 0 입력시, Identity 가 됩니다.

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)

ttans1 이 x1,y1,z1,a1,b1,c1 의 값이 할당된 trans 변수로 생성된다.

Result:

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

Translate – Trans 위치 변수를 x, y, z 값만큼 이동합니다.

Description:

첫 번째 인자인 Trans 변수의 위치에서 두 번째 인자인 X , 세 번째 인자인 Y, 네 번째 인자인 Z 좌표를 ±하여 Trans 에 할당합니다

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)

newposition 에 nowposition 부터 dx1, dy1, dz1 만큼 이동한 값이 할당됩니다.

Result:

newposition : 20, 30, 20

Value – String data를 Number로 변환하여 할당합니다.

Description:

String data 를 Number data 로 변환하여 할당합니다.

Syntax:

value1 = value($value2)

Return value: value1

Data type : Number

Arguments: $value2

Data type : String

Example1:

$stringnum = "12233334444"

valuenum = Value($stringnum)

valuenum 에 $stringnum 의 값이 Number 로 변환되어 입력됩니다.

Result:

valuenum : 1223334444

13. Instruction Reference

ABOVE/BELOW – Robot의 elbow 형상 설정

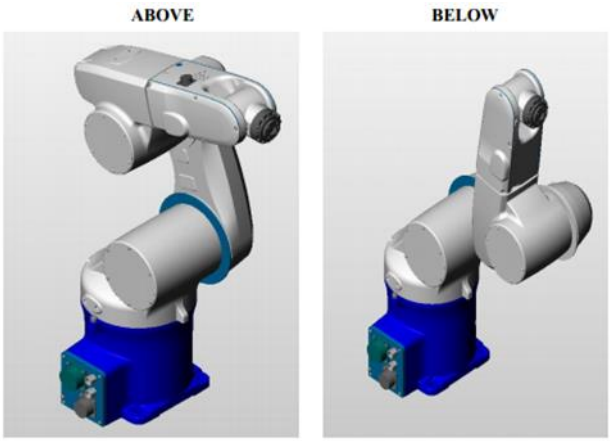
Description:

6 축 Robot 에서 특정 경계 위치에 있을 때 동작 방향을 결정 해야합니다.
이 때에 Robot 의 elobow 를 윗면 기준으로 움직일 것인지 아랫면 기준으로 움직일 것인지 형상을 결정할 수 있습니다, ABOVE 는 윗면이 위로 가고, BELOW 는 아랫면이 위로 가는 형상이 됩니다.
Joint2, 3 으로 결정합니다.

Syntax:

ABOVE

BELOW



Accel/Decel – 가속도와 감속도를 설정합니다.

Description:

Robot 의 하중이 무거운 물체를 잡고 움직이면, Default 가감속으로는 동작에 무리가 발생할 수 있습니다, 이 때 무리가 될 수 있는 구간에 가감속을 변경 하여 유연하게 동작할 수 있게 설정할 수 있습니다

MAX 속도는 ROBOT_CONFIG 에 정의 되어 있습니다.

Fixed 를 사용하지 않으면, 다음 모션 명령에만 적용됩니다.

Syntax:

Accel aspeed [Fixed]

Decel dspeed [Fixed]

Argumen ts: aspeed, dspeed, Fixed

Data type : Value Number, Value Number, Command

Value Number 는 Max 가감속의 % 단위입니다, 입력 범위는 0.01 ~100%

Example:

Accel 10

Decel 20

MoveJ #p1

MoveJ #p2

Accel 20 Fixed

Decel 20 Fixed

MoveJ #p3

MoveJ #p4

#p1 동작 시 Accel 10, Decel 20 적용, #p2 동작 시에는 default 가감속 적용,

#p3, #p4 동작 시 Accel 20, Decel 20 은 계속 적용됩니다.

Acuracy – 연속 동작의 정밀도를 설정합니다.

Description:

Robot 이 position1 -> 2 -> 3 로 이동할 때 Accuracy 값을 높게 설정하면, 로봇이 position1 에서 position2 로 끝까지 이동하지 않고 설정한 값 만큼 간격을 두고 부드럽게 지나쳐 position3 으로 이동합니다.
감속 구간과 가속 구간의 합성으로 동작이 결정되며 설정한 가 감속 구간의 거리 내에서 설정 값을 설정 해야 합니다, 단위는 거리 값이므로 mm 입니다

Syntax:

Accuracy distance1 Fixed

Arguments: distance1, Fixed

Data type : value Number, Command

Example:

```
MoveL p1
Accuracy 20
MoveL p2
MoveL p3
MoveL p4
Accuracy 10 Fixed
MoveL p5
MoveL p6
MoveL p7
```

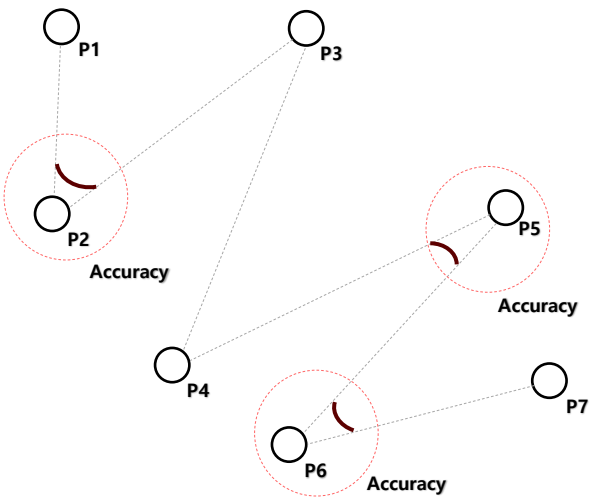


Figure 8 Accuracy 모션 동작

p1 으로 이동, Accuracy 20 설정, p2 로 이동할 때 Accuracy 적용,
p3 로 Default Accuracy 이동, p4 로 Default Accuracy 이동,
Accracy 10 Fixed 설정 고정 p5 로 이동할 때 Accuracy 적용,
p6 로 이동할 때 Accuracy 적용, p7 까지 부드럽게 이동 후 동작 완료

ALIGN – Tool의 z 방향을 가장 가까운 base의 좌표축과 맞춥니다

Description:

Tool 의 z 방향을 가장 가까운 base 의 좌표축과 맞춥니다.

Syntax:

Align

Arguments: Align

Align 을 사용하면 tool 의 z 축을 base 좌표축과 맞추기 위해 이동합니다

Example:

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

MoveJ #joint1

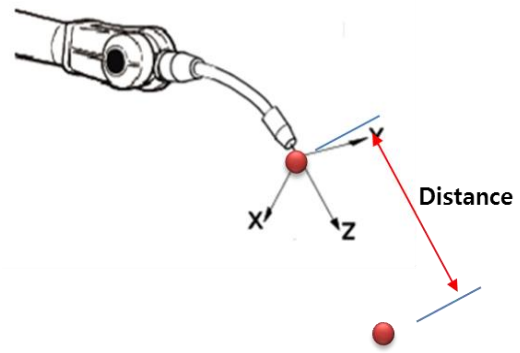
Align

MoveJ 로 로봇을 움직인 후 Align 을 사용하여 Tool 의 z 방향을 가장 가까운 base 의 좌표축과 맞춥니다.

ApproJ – Robot이 Tool 좌표계의 -Z방향으로 입력한 만큼 이동합니다

Description:

Robot 이 입력한 좌표점 P1 으로 빠르게 이동하기 위하여 ApproJ 명령을 사용합니다. P1 을 기준으로 Tool 좌표계의 -Z 축 방향으로 설정한 거리 만큼 떨어진 위치로 이동합니다.



Syntax:

ApproJ #joint1, distance1

ApproJ trans1, distance1

Arguments: #joint1 or trans1, distance1

Data type : #Joint or Trans, Number

첫 번째 인자는 기준 좌표, 두 번째 인자는 Tool 좌표계에서 -Z 축 방향으로 기준점으로부터 떨어진 거리

Example:

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

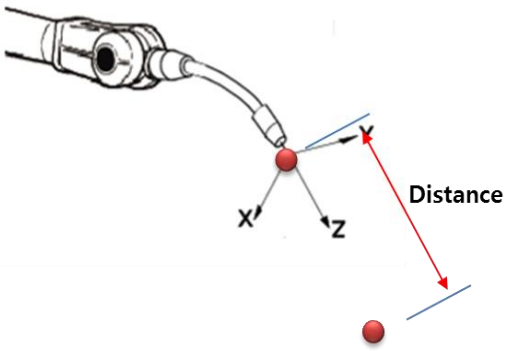
ApproJ #joint1, 10

#joint1 위치에서 tool 좌표계 기준으로 -Z 축으로 10 만큼떨어진 위치로 이동합니다.

ApproL – Robot이 Tool 좌표계의 Z방향으로 입력한 만큼 직선보간 이동합니다

Description:

Robot 이 입력한 좌표점 P1 으로 빠르게 이동하기 위하여 ApproJ 명령을 사용합니다. P1 을 기준으로 Tool 좌표계의 -Z 축 방향으로 설정한 거리 만큼 떨어진 위치로 직선보간 합니다.



Syntax:

ApproL #joint1, distance1

ApproL trans1, distance1

Arguments: #joint1 or trans1, distance1

Data type : #Joint or Trans, Number

첫 번째 인자는 기준 좌표, 두 번째 인자는 Tool 좌표계에서 -Z 축 방향으로 기준점으로부터 떨어진 거리

Example:

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

ApproL #joint1, 10

#joint1 위치에서 tool 좌표계의 -Z 축 방향으로 10mm 떨어진 점으로 직선보간 이동합니다.

Brake – 동작을 멈추고 다음 step을 진행합니다.

Description:

Robot 이 동작 중 Brake 를 만나면 동작을 멈추고 다음 step 을 실행합니다.

Syntax:

Brake

Arguments:

Data type : Command

Example:

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

동작 중 digital input 1007 에 Signal 이 발생 하면 Brake 가 되어 현재 동작을 멈추고 다음 step 을 실행한다.

Break – CP 동작을 끊어, 정확한 위치로 가게 합니다

Description:

Robot 이 이동할 때 Accuracy 를 Skip 하고 정확한 위치에 도달할 때까지 다음 Step 으로 넘어가지 않고 대기하고 있다가 정확한 위치에 Robot 이 이동하였을 때 다음 Step 을 진행 시키는 명령어입니다.

Syntax:

Break

Arguments:

Data type : Command

Example:

MoveL p1

MoveL p2

MoveL p3

Break

MoveL p4

MoveL p5

P1 ~ p3 까지 default Accuracy 값이 적용되어 부드럽게 이동하지만 P3 에서 p4 로 이동하는 구간은 Accuracy 값이 Skip 되어 정확하게 p4 로 이동하게 됩니다, p4 에서 p5 까지는 Accuracy 값이 적용 됩니다.

Call – Macro Program을 호출하여 실행합니다.

Description:

작성한 Macro Program 을 호출하고 바로 실행합니다.

Syntax:

Call program

Arguments: prgoram

Data type : string

프로그램 명칭을 직접 입력합니다.

Example1:

Call examprg1

Call examprg2

examprg1 이 call 되어 먼저 실행됩니다, 프로그램 동작이 완료되면 다음 step 에 있는 examprg2 가 실행됩니다.

Delay – Robot의 동작을 정지 시킵니다.

Description:

Robot 동작을 정지 시킵니다. 목표치 제어를 입력한 시간동한 계속 수행하기 때문에, 정확한 제어 위치에 도달할 때까지 여유시간이 필요할 때 사용합니다. Delay 는 모션 명령으로 DelayTime + MoveJ 입니다. 즉 설정한 시간만큼 지정된 위치로 MoveJ 명령이 나갑니다.

Syntax:

Delay time1

Arguments: time1

Data type : Number

원하는 delay 시간을 입력한다 단위는 sec

Example:

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

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

MoveJ #joint1

Delay 10

MoveJ #joint2
```

#joint1 까지 이동 후 10 초 동안 MoveJ 명령으로 정확한 #joint1 로 이동합니다.

10 초후 #joint2 위치로 이동합니다.

DelayTime – 다음 명령까지 대기합니다.

Description:

설정한 시간만큼 다음 명령까지 대기합니다. 지정된 목적지까지 정확하게 이동하기 위하여 다음 Command 명령이 까지 설정 시간만큼 대기합니다.

Syntax:

DelayTime time1

Arguments: time1

Data type : Number

원하는 delay 시간을 입력합니다. 단위는 sec

Example:

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

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

Point #joint3 = Joint(0,150,0)

MoveJ #joint1

DelayTime 0.1

MoveJ #joint2

MoveJ #joint3
```

#joint1 위치로 이동 합니다.

#joint2 위치로 이동합니다. 정확한 #joint2 위치까지 이동하기 위하여 그 다음 명령(MoveJ #joint3)이 올때까지 0.1 초 대기합니다..

DelayDO – 설정한 시간 후에 Digital Output을 출력합니다.

Description:

출력할 Digital Ouput number 와 시간을 설정 후 출력합니다

동작 시 시간이 카운트 되며 설정 시간이 종료되면 DO 출력합니다. .

출력된 DO 는 Reset 전까지 유지됩니다.

Syntax:

DelayDO output1, time1

Arguments: output1, time1

Data type : Digital Output Number, value Number

DO 는 1~128 까지 설정하여 사용할 수 있습니다.

Value Number 에 출력 시간을 설정합니다. 단위는 sec 입니다.

Example:

DelayDO 1, 0.5

DelayDO 2, 1

WaitTime 1

SetDO -1

SetDO -2

DO 신호 1 번 0.5 sec 후에 On 되고, DO 신호 2 번은 1sec 후에 On 됩니다,

1 초 동안 대기 후 SerDO -1, -2 되어 DO 출력이 모두 Off 됩니다.

DepartJ – Robot이 Tool 좌표계의 –Z방향으로 입력한 만큼 이동합니다

Description:

Robot 이 Tool 좌표계의 -Z 방향으로 입력한 만큼 이동합니다.

Syntax:

DepartJ distance1

DepartJ distance1

Arguments: distance1

Data type : Number

Tool 의 Z 방향으로 이동할 거리

Example:

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

MoveJ #joint1

DepartJ 10

#joint1 위치에서 tool 좌표계로 Z 축 -10 만큼을 이동합니다.

DepartL – Robot이 Tool 좌표계의 –Z방향으로 입력한 만큼 직선보간 이동합니다

Description:

Robot 은 Tool 좌표계의 -Z 방향으로 입력한 만큼 직선보간 이동합니다.

Syntax:

DepartL distance1

DepartL distance1

Arguments: distance1

Data type : Number

Tool 의 Z 축 방향으로 이동할 거리

Example:

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

MoveJ #joint1

DepartL 10

#joint1 위치에서 tool 좌표계로 Z 축 -10 만큼을 직선보간 이동합니다.

Drive – 설정한 축이 상대이동 합니다

Description:

현재 위치 기준으로 지정한 Joint 만 입력한 수치만큼 상대이동을 합니다..

Syntax:

Drive delta_joint, value1

Arguments: delta_joint, value1

Data type : Delta Joint Number, value Number

이동하기 원하는 Joint Number 와 이동하기 원하는 value Number 를 설정합니다.

Example:

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

MoveJ #joint1

Drive 1, 20

Drive 2, 10

Drive 3, 10

MoveJ 하여 #joint1 의 위치로 이동 합니다

첫 번째 Drive 에서 Joint1 을 20[mm or degree] 이동 합니다

두 번째 Drive 에서 Joint2 를 10[mm or degree] 이동 합니다

세 번째 Drive 에서 Joint3 을 10[mm or degree] 이동 합니다.

FMoveL – 고정된 FTool을 균일한 거리를 유지하며 동작합니다.

Description:

지정한 FTool(Fixed Tool)에 대해서 균일한 거리를 유지하며 동작합니다.

Syntax:

FMoveL #jvar1

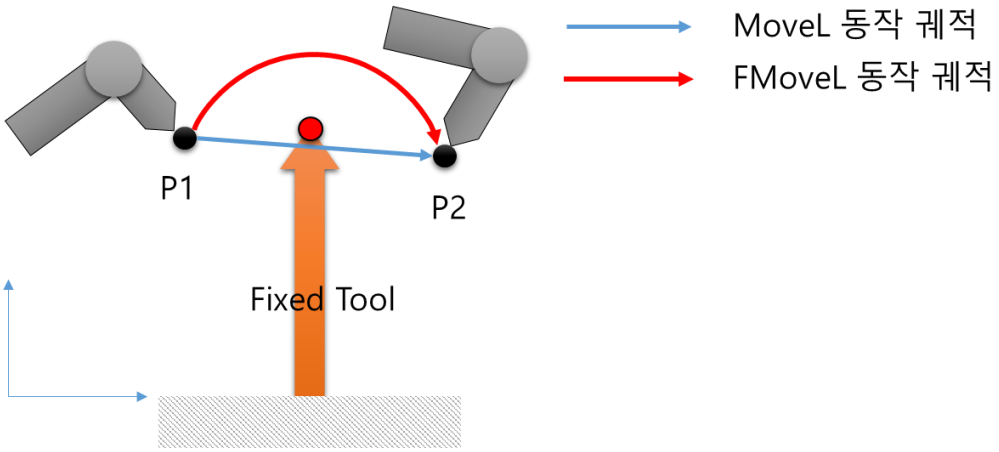
FMoveL tvar1

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
```



FTool – FTool(Fixed Tool)고정된 Tool을 설정합니다.

Description:

FTool 좌표계를 설정하는 명령어 입니다.

Syntax:

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

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

Data type : Number

Trans 좌표 값을 입력. 자세를 표현하는 euler angle 값은 0, 0, 0 입력시, Identity 가 됩니다.

Example:

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

HALT – 프로그램을 중단시킵니다.

Description:

원하는 위치에서 프로그램을 중단시킵니다.

Syntax:

HALT

프로그램에서 원하는 step 에서 프로그램을 중단시킵니다.

Example:

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

MoveJ 로 #p1 으로 이동합니다

HALT 명령어로 프로그램을 중단합니다.

프로그램이 중단되어 #p2 로 이동하지 않습니다.

HOME/HOME2 – Home 위치로 이동합니다.

Description:

지정한 Home 위치로 이동합니다, home 과 home2 를 지정할 수 있습니다.

Syntax:

Home

Home 2

Arguments: HOME or HOME 2

지정한 위치로 이동한다, 지정한 위치는 home position 이 됩니다.

Home position 을 지정하는 방법은 setting 의 Set Home 을 사용합니다.

Example:

HOME

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

MoveJ #joint1

HOME 2

HOME 를 사용하여 처음에는 HOME 에 지정한 좌표로 이동합니다

MoveJ 로 로봇을 움직인 후 다시 HOME2 위치로 이동하여 동작을 완료합니다.

IncJ – Joint가 상대이동 합니다.

Description:

현재 위치 기준으로 입력한 수치만큼 지정한 Joint 가 Joint 보간으로 이동 합니다.
0 를 입력 하면 해당 Joint 는 이동 하지 않습니다.

Syntax:

IncJ delta_joint1, delta_joint2, ...

Arguments: delta_joint1, dalta_joint2

Data type : Delta Joint Number

#Joint 에 포함 되어있는 각 delta joint 의 좌표 값

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

#p1 위치로 이동한 후, 처음에는 delta joint1 이 10 이동 그 다음 delta joint2 가 10 이동, 마지막에 delta joint3 가 10 이동 합니다.

IncL – Base좌표계에 대한 직선보간으로 상대이동 합니다.

Description:

현재 위치 기준으로 입력한 수치만큼 직선보간 이동을 합니다. 좌표축 기준은 Base 좌표계입니다. 즉, Target Trans 는 아래의 식과 동일합니다.

Ptarget = Pcurrent + Pdelta

Syntax:

IncL delta_x, delta_y, delta_z, ...

Arguments: delta_x, delta_y, delta_z, ...

Data type : Delta Trans matrix Number

Trans 에 포함 되어있는 각 delta 축의 좌표 값

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

MoveJ #p1 에 해당하는 초기 위치로 이동합니다. 그다음 IncJ 명령어를 통해, 차례차례 현재 위치에서 delta x, 10 이동하고, 그 다음 delta y 를 20 이동, 그리고, 마지막에 delta z 가 30 이동 합니다. 이동시에는 직선 보간으로 이동하게 됩니다.

IncT – Tool 기준으로 상대이동 합니다.

Description:

현재 위치에서 Tool 좌표계 기준으로 입력한 수치만큼 상대이동을 합니다..

Syntax:

IncT delta_tx, delta_ty, delta_tz, ...

Arguments: delta_tx, delta_ty, delta_tz, ...

Data type : Delta Trans matrix Number

Tool 기준의 Trans matrix x, y, z

Example:

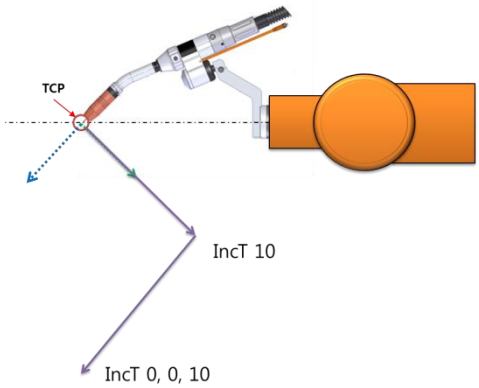
MoveJ #p1

IncT delta_tx, 0, 0

IncT 0, delta_ty, 0

IncT 0, 0, delta_tz

#p1 위치로 이동한 후, Tool 좌표 기준으로 , 처음에는 delta tx 가 10 이동 그 다음 delta ty 가 10 이동 마지막에 delta tz 가 10 이동 합니다.



LEFTY/RIGHTY – 로봇의 arm 형상 설정

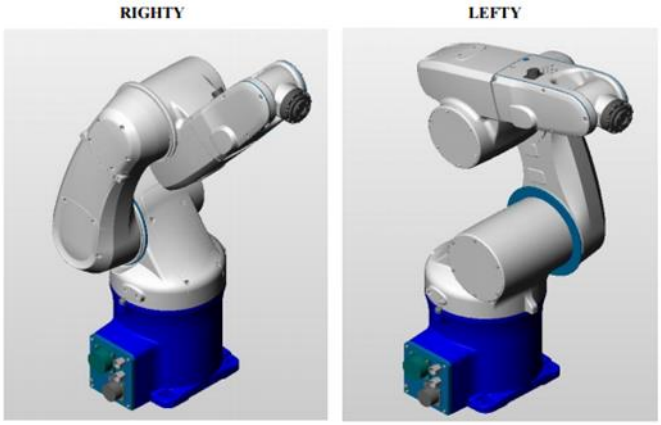
Description:

6 축 Robot 에서 특정 경계 위치에 있을 때 동작 방향을 결정해야 합니다.
방향이 결정되어 있지 않을 때 전혀 다른 방향으로 이동하게 됩니다
로봇의 형상에 오른팔의 형상을 하고 있으면 RIGHTY, 왼팔의 형상을 하고 있으면 LEFYT 를 사용하여 형상을 결정합니다, 만약 로봇이 대칭이면 형상 결정에 의미가 없습니다, Joint 1, 2, 3 으로 결정합니다.

Syntax:

LEFTY

RIGHTY



MakeStr – String 변수를 생성합니다.

Description:

다양한 형태의 변수를 String 으로 만들어 줍니다.

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]

#Here 로 현재 Joint 위치를 얻는다.

각축의 Joint 값을 얻는다.

MakeStr 명령으로 $msgsend 를 생성한다.

MoveC – 원호보간으로 동작합니다.

Description:

경유 지점과 종착 위치를 입력하여 원호보간으로 동작합니다.

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

로봇이 원호를 돌기 전 trans1 의 위치에 있는다

Trans1 의 위치부터 시작하여 passtrans 위치를 경유하고 endtrans 위치에서 동작이 완료된다.

MoveH – Singular 구간의 통과합니다(Serial6만 해당).

Description:

로봇이 Singular 구간을 통과할 때 모터의 속도가 급격히 증가되어 Over speed Limit 에러가 발생하고 로봇이 정지하게 됩니다. 이 현상을 회피하는 방법으로 로봇의 speed 를 줄여 singular 구간을 통과하게 하는 명령어 입니다. 하지만 속도의 조절만으로 Singular 구간을 회피하기 때문에 경우에 따라 작동이 완벽하게 되지 않을 수도 있습니다.

MoveH 명령어를 사용하려면 corecon.conf 에 다음 옵션을 추가하여야 합니다.

Teach mode singular 회피:

TEACH_SINGULAR = TRUE

TEACH_SINGULAR_SPEED_LIMIT = 0.001~1.0 (1.0 이면, teach mode 의 max speed limit 까지 허용)

Repeat mode singular 회피:

REPEAT_SINGULAR = TRUE

REPEAT_SINGULAR_SPEED_LIMIT = 0.001 이상(1.0 이면, Repeat mode max speed 로 설정됩니다. 보통 정격 속도의 2 배)

따라서, AXIS Spec 항목인 AXIS_MAXRPM, AXS_RPM 등의 설정치를 같이 조정할 필요도 있습니다.

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
```

MoveJ – 입력된 위치로 Joint 이동합니다.

Description:

Joint 혹은 Trans 변수에 입력된 위치로 Joint 이동합니다. .

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
```

MoveL – 입력된 위치로 직선보간 이동합니다.

Description:

Joint 혹은 Trans 변수에 입력된 위치로 직선보간 이동합니다. .

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
```

MoveX – 신호를 모니터링하며 직선 보간 이동합니다

Description:

시작 위치에서 직선보간 이동 중 지정한 DI 에서 Signal 을 받으면 현재 이동을 정지한 후 Signal 을 받은 위치에서 다음 Step 을 실행합니다.

Syntax:

MoveX trans1, 1007

Arguments: #joint1, 1007

Data type : trans1, Input Signal Number

첫 번째 인자는 Joint 변수 입력, 두 번째 인자는 DI 번호를 입력합니다.

Example:

Point trans1 = trans(10,10,0)

Point trans2 = trans(150,150,0)

MoveX trans1, 1007

MoveL trans2

MoveX 명령을 받고 trans1 으로 이동 중 Robot 이 목적 위치로 이동 중에

DI 1007 에 입력 신호를 받으면 MoveX 명령을 중단하고 즉시 다음 Step 인

MoveL 명령을 실행하여 trans2 위치로 이동한다.

MoveXJ – 신호를 모니터링하며 Joint 이동합니다

Description:

시작 위치에서 Joint 이동 중 지정한 DI 에서 Signal 을 받으면 현재 이동을 정지한 후 Signal 을 받은 위치에서 다음 Step 을 실행합니다.

Syntax:

MoveXJ #joint1, 1007

Arguments: #joint1, 1007

Data type : #Joint, Input Signal Number

첫 번째 인자는 Joint 변수 입력, 두 번째 인자는 DI 번호를 입력합니다.

Example:

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

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

MoveXJ #joint1, 1007

MoveJ #joint2

MoveXJ 명령을 받고 #joint1 으로 이동 중 Robot 이 목적 위치로 이동 중에

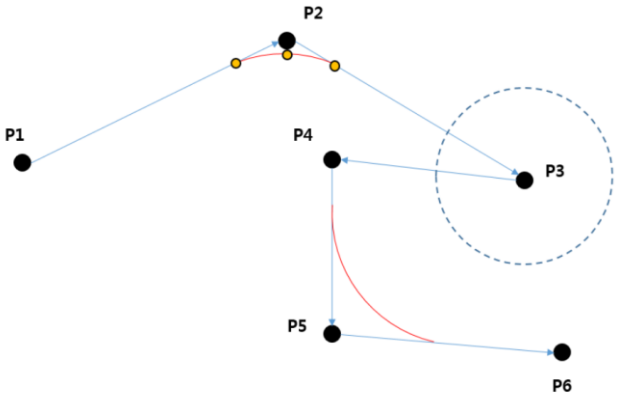
DI 1007 에 입력 신호를 받으면 MoveXJ 명령을 중단하고 즉시 다음 Step 인

MoveJ 명령을 실행하여 #joint2 위치로 이동한다.

MovePi, MoveP – 여러 개의 Point를 하나의 선으로 이어 등속으로 이동한다.

Description:

여러 개의 점들을 teaching 한 후 이동할 때 등속으로 이동이 필요한 경우 사용되는 명령어로 등속의 시작 점과 끝점까지 하나의 선으로 이동하는 것과 같이 움직이게 된다.



그림과 같이 P1 ~ P6 까지 구간을 등속으로 움직이고, Arc 반지름을 지정하면, 연결부는 ARC motion 을 하게 된다.

Syntax:

MovePi joint1[trans 좌표], R[반지름] --- 시작점과 중간 점

MoveP joint2[trans 좌표], R[반지름] --- 시작점과 마지막 점

Arguments: Joint1, Joint2, R

Data type : Joint1, Joint2 = Trans Point

R = Number

Example:

```
MoveJ P1
MovePi P2, 10    ;  반지름 10
MovePi p3, 0     ; 반지름 없음, Arc 없음
MovePi p4, 0
MovePi p5, 20
MoveP p6, 0      ; 마지막 점은 반지름이 적용되지 않기 때문에, 0 을 입력해 준다.
```

MSig – Move 동작 중 IO Signal을 발생시킨다.

Description:

Move 동작명령 전에 MSig 를 실행 시키면 Move 할 때 MSig 설정에 따라 IO 동작을 실행합니다.

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, Number
Signal 은 Dout 의 number 를 입력 operation 에 의해 On/Off 됩니다.
+는 On, -는 Off 입니다.
option : MSTIME(시간 sec 기준), MSDIST(거리 mm 기준), MSPERC(백분율 % 기준) 으로 On/Off 를 설정하는 option 입니다.
value : option 에 따라 시간, 거리, 백분율(%) 값을 설정합니다.

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 1 초 후 on, 21 2 초 후 on 설정, MoveJ 동작 시 설정에 따라 동작.
Dout 22 100mm 이동 후 on, 20 도착 100 전에 off 설정, MoveJ 동작 시 동작
Dout 23 30% 이동 후 On, 21 도착 30% 전에 off 설정, MoveJ 동작 시 동작
Reset 출력 초기화.

●**주의** MSig 명령어를 사용할때 순서대로 MSig 명령을 check 하기 때문에 유의하시기 바랍니다.

Ex:

MSig -15, MSDIST, -0.1

MSig 15, MSDIST, 0.1

MoveJ #joint1

위 순서대로 프로그램을 하게되면 마지막 끝점에서 DO15 번이 OFF 가 안됩니다.

OverDI – SetDI 동작을 On/Off합니다.

Description:

Macro 프로그램 동작 중에 SetDI 조작을 On/Off 할 수 있습니다.

Syntax:

OverDI On/Off

Example:

```
OverDI ON
SetDI 1002 ; DIn 2 On
WaitTime 1
SetDI -1002 ; DIn 2 Off
OverDI OFF
```

PrefetchSig – PREFETCH_SIGNAL명령을 On/Off 합니다.

Description:

로봇의 모션 명령 후의 DI, DO 출력 타이밍을 설정합니다.

PrefetchSig On 이면 로봇 모션이 시작할 때 DI, DO 가 출력되며, PrefetchSig Off 일 때 모션 종료 후 출력이 나가게 됩니다.

Syntax:

PrefetchSig On/Off

Example:

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

MoveJ #p1 모션과 함께 SetDO 10 이 동작합니다.

MoveJ #p2 모션이 끝난 지점(#p2)에서 SetDO -10 이 동작합니다.

Reset – 모든 출력을 Off 합니다.

Description:

DO, AO 에서 출력되는 모든 신호를 off 하는 명령어입니다.

Syntax:

Reset

Arguments:

Data type : Command

Example:

SetDO 1

SetDO 2

WaitTime 1

Reset

Digital output 신호 1, 2 번 출력, 1 초 대기 후 모두 off 합니다.

RunMask – 실행 시에만 Signal을 내보내는 옵션.

Description:

IO 신호는 프로그램을 정지할 경우, 계속 내 보내거나, 강제로 Off 할 필요가 있습니다. 프로그램 정지시 강제로 Off 하고자 할 때, RunMask 로 출력을 조정할 Output 을 설정합니다.

설정된 DO 는 Macro 가 정시 시 Off 됩니다. Macro 의 동작 status 를 별도로 출력하여 확인할 수 있습니다.

Syntax:

RunMask startdo, count1

Arguments: startdo, count1

Data type : Digital Output Number, value Number

DO 는 1~128 까지 설정하여 사용할 수 있습니다.

Value Number 는 출력할 DO 의 수량을 설정합니다.

프로그램의 startdo 에 DO10 을 설정하면 count 는 11, 12, 13 번 순으로 되어 설정됩니다.

Example:

RunMask 6, 4

MoveJ #p1

MoveJ #p2

초기 RunMask 가 선언되면서 DO6 부터 7, 8, 9 총 4 개의 DO 가 선언됩니다

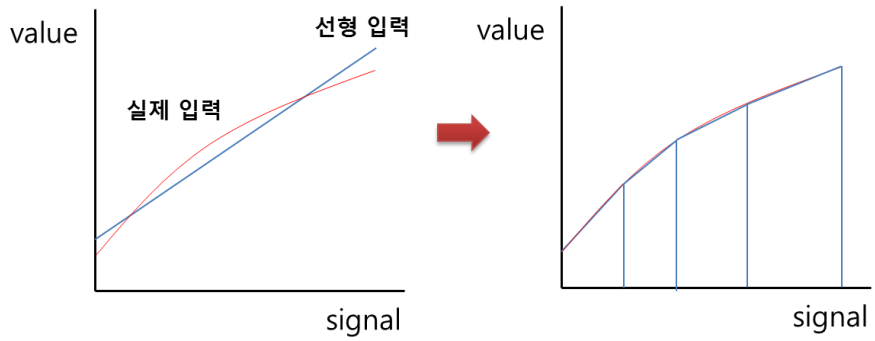
Macro 가 Running 상태일 때 DO6~9 번은 출력이 됩니다 Hold 되거나

Macro 가 끝났을 때 출력이 Off 됩니다.

SetAICalib – Analog input channel의 calibration table을 설정

Description:

Analog 의 입력 시그널 레벨과, 실제 물리적 값의 calibration table 을 설정합니다. Analog 입력신호와 물리적 값은 아래 그림과 같이 선형적으로 변화 하지 않는 경우가 많습니다. 이러한 경우, calibration table 을 사용하여, 소규모의 선형 구간으로 분할하여, 출력의 정밀도를 향상 시킬 수 있습니다. 반드시 최소, 최대값은 입력하여야 합니다.



Syntax:

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

Arguments: channel#, signal, value

Channel# : numeric 1~128 까지 설정하여 사용할 수 있으나, 실제 입력은 설치된 device channel 개수에 제한됩니다

Signal, value : calibration table 을 구성하는 signal, value 의 쌍을 입력합니다. 최대 입력은 20 개의 쌍까지 입력할 수 있습니다.

Example:

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

아래와 같은 calibration table 을 정의합니다.

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

Temp = AIN(1)

만일 1 번 채널의 Analog 입력 전압이 6v 가 되면, temp 는 300 이 되게 됩니다.

SetAO – Analog Output을 출력합니다.

Description:

출력할 Analog output channel 을 설정하여 출력합니다

출력할 값은 신호의 전압이나 전류값이 아닌, calibration table 에 정의된 실제 표현 값으로 설정합니다. Calibration table 정의는 SetAOCalib 명령어를 참고합니다.

Syntax:

SetAO channel#, value

Arguments: channel#, value

Channel# : numeric 1~128 까지 설정하여 사용할 수 있으나, 실제 출력은 설치된 device channel 개수에 제한됩니다.

Value : 출력할 analog 값

Example:

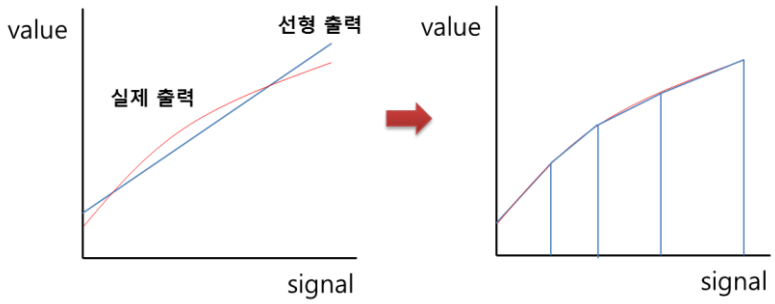
```
SetAO 1,  30
WaitTime 1
SetDO 1, 0
```

1 번 채널의 analog 값을 30 으로 설정 후, 0 으로 설정합니다. 설정값은 calibration table 을 참조하여, 신호값으로 변환되어 출력됩니다.

SetAOCalib – Analog output channel의 calibration table을 설정

Description:

Analog 의 출력 시그널 레벨과, 실제 물리적 값의 calibration table 을 설정합니다. Analog 출력신호와 물리적 값은 아래 그림과 같이 선형적으로 변화 하지 않는 경우가 많습니다. 이러한 경우, calibration table 을 사용하여, 소규모의 선형 구간으로 분할하여, 출력의 정밀도를 향상 시킬 수 있습니다. 반드시 최소, 최대값은 입력하여야 합니다.



Syntax:

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

Arguments: channel#, signal, value

Channel# : numeric 1~128 까지 설정하여 사용할 수 있으나, 실제 출력은 설치된 device channel 개수에 제한됩니다

Signal, value : calibration table 을 구성하는 signal, value 의 쌍을 입력합니다. 최대 입력은 20 개의 쌍까지 입력할 수 있습니다.

Example:

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

아래와 같은 calibration table 을 정의합니다.

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

SetDO – Digital Output을 출력합니다.

Description:

출력할 Digital Output number 를 설정하여 출력합니다

DO 를 출력할 때 출력 number 에 '-' 부호를 넣으면 해당 DO 는 Reset 됩니다.

Syntax:

SetDO output1

Arguments: output1

Data type : Digital Output Number

DO 는 1~128 까지 설정하여 사용할 수 있습니다.

Example:

SetDO 1

SetDO 2

WaitTime 1

SetDO -1

Digital output 신호 1, 2 번 출력, 1 초 대기 후 1 번만 off 합니다.

SetJ – Joint 변수값을 설정합니다.

Description:

Joint 변수값을 설정합니다.

Syntax:

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

Arguments: #jval, j1 ~ 6

#jval : joint 변수

j1~6 은 Joint1~ 6 번째와 같습니다.

Example:

Point #jval = Joint(0,0,0,0,0,0)

SetJ #jval, 10, 10, 10, 10, 10, 10

MoveJ #jval

Joint 변수 #jval 를 생성합니다.

#jval 변수 j1~j6 을 10 으로 설정합니다.

설정된 #jval(Joint(10,10,10,10,10,10))로 Joint 이동합니다.

SetP – Trans 변수값을 설정합니다.

Description:

Trans 변수값을 설정합니다.

Syntax:

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

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

tval : trans 변수

Trans 좌표 값을 입력. 자세를 표현하는 euler angle 값은 0, 0, 0 입력시, Identity 가 됩니다.

Example:

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

trans 변수 tval 를 생성합니다.

tval 변수 x, y, z, a, b, c 를 각각 1280, 0, 1570, 0, 90, 0 으로 설정합니다.

설정된 tval(trans(1280,0,1570,0,90,0))로 Joint 이동합니다.

SClose – serial통신을 닫아 줍니다.

Description:

serial 을 닫아줍니다 .

Syntax:

SClose sid

Arguments: sid

sid: serial 통신 id

Example:

```
sid = 0
SClose sid
```

SOpen – serial통신을 열어줍니다.

Description:

serial 을 열어줍니다 .

Syntax:

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

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

sid: serial 통신 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

SRead – Serial 데이터를 읽어옵니다.

Description:

Serial 데이터를 읽어옵니다..

Syntax:

SRead sid, $str_read, ret_flag

Arguments: sid, $str_read, ret_flag

sid: serial 통신 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

SRead2 – Serial data를 구간 선택하여 읽어옵니다.

Description:

Serial data 를 구간 선택하여 읽어옵니다.

Syntax:

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

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

sid: serial 통신 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
```

SWrite – Serial data를 Write합니다.

Description:

Serial data 를

Syntax:

SWrite sid, $msgwrite, ret_flag

Arguments: sid, $msgwrite, ret_flag

sid: serial 통신 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
```

SINGLE/DOUBLE – 6축의 각도에 따라 회전 범위를 결정합니다.

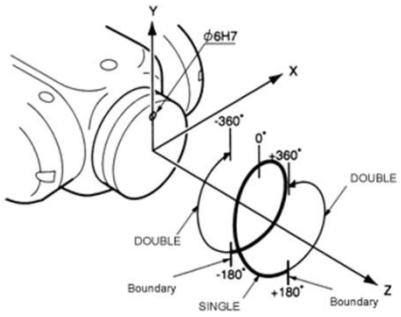
Description:

만약 6 축을 1 바퀴 회전할 수 있는 Tool 이나 형상을 사용한다면 SINGLE
2 바퀴 회전할 수 있는 Tool 이나 형상을 사용한다면 DOUBLE 을 선택합니다.
Joint6 으로 결정합니다.

Syntax:

SINGLE

DOUBLE



Smooth – S-curve motion을 구현합니다.

Description:

Robot 이 동작할 때 S-curve motion 을 구현하여 motion 을 부드럽게 만듭니다.

Syntax:

Smooth TRUE/FALSE, [jerk time]

Arguments: speed1, Fixed

Data type : **Bool,** Value Number

Bool : TRUE 를 입력하면 S-curve motion 을 on 합니다, FALSE 시 off 합니다.

Value : S-curve motion 의 jerk 값을 입력하여 S-curve motion 을 설정합니다.

합니다.

Example:

```
Smooth TRUE, 0.3
MoveL pt100        ; pt100 위치로 이동할 때 S-curve motion 이 적용됩니다.
Smooth FALSE, 0
MoveL pt200        ; pt200 위치로 이동할 때 trapezoidal motion 이 적용됩니다.
```

Stable – 로봇이 설정시간 동안 지정된 위치까지 계속 움직입니다.

Description:

Delay 와 Stable 은 motion 명령어 입니다. Move 명령어 뒤에 작성된 Delay 와 Stable 명령어는 움직임이 끝날 때까지 기다립니다. Delay 는 단지 이전의 움직임이 끝난 후에 기다리는 작업을 수행하지만, Stable 은 주어진 시간동안 target 위치로 움직이라는 명령을 계속 실행합니다(MoveJ 현재 위치 + stable 시간). 이 작업은 robot 움직임을 안정화 시키고 정확도를 증가시킵니다. motion 이 아닌 명령어는 Delay 와 Stable 명령어 뒤에 실행됩니다. non-motion 명령어는 Move 명령어와 같이 즉시 실행됩니다.

Syntax:

Stable time1

Arguments: time1

Data type : Value Number

원하는 delay 시간을 입력한다. 단위는 sec

Example:

MoveJ #joint1

Stable 1

MoveJ #joint2

로봇이 #joint1 로 이동 후 1 초동안 MoveJ 명령으로 #joint1 위치까지 계속 이동합니다. 1 초안에 로봇이 지정된 오차 범위를 벗어나면 Wait 시간은 다시 1 초로 변경되고 #joint2 위치로 이동하지 않습니다. 정확한 위치에 도착할 때까지 무한 이동하기 때문에 사용하실 때 주의하여야 합니다.

StableTime – 모션 명령어에 Stable 기능을 사용합니다.

Description:

모션 명령어 Stable 과 달리 StableTime 은 Non-Motion 명령어입니다. 모션 명령어 바로 위에 사용되며 모션 명령어와 즉시 실행됩니다. 모션 명령어에 Stable 기능을 사용하기 위한 명령어입니다.

Syntax:

StableTime time1

Arguments: time1

Data type : Value Number

원하는 delay 시간을 입력한다. 단위는 sec

Example:

MoveJ #joint1

StableTime 1

MoveJ #joint2

MoveJ #joint3

로봇이 #joint1 로 이동 합니다.

#joint2 위치로 이동합니다. 1 초동안 MoveJ 명령으로 #joint2 위치까지 계속 이동합니다. 1 초안에 로봇이 지정된 오차 범위를 벗어나면 Wait 시간은 다시 1 초로 변경되고 #joint3 위치로 이동하지 않습니다. 정확한 위치에 도착할 때까지 무한 이동하기 때문에 사용하실 때 주의하여야 합니다.

SPEED – 동작 속도를 설정합니다.

Description:

Robot 의 동작 속도를 설정합니다, Fixed 를 설정하면 이 후 모든 동작에서의 속도가 지정된 속도로 설정 되고, Fixed 를 사용하지 않으면, 다음 모션 명령의 속도만 변경 됩니다.

Syntax:

Speed speed1 [Fixed]

Arguments: speed1, Fixed

Data type : Value Number

speed1 을 입력 하고 Fixed 를 입력하지 않으면 다음 모션 명령에만 speed1 의 값이 적용됩니다.

speed1 은 mm/sec, mm/min 와 % 세 종류가 있습니다.
mm/sec, mm/min 으로 설정 시에만 mm/s, mm/min 를 입력 합니다.

Example:

```
Speed 10
MoveJ #joint1
MoveJ #joint2
Speed 20mm/s Fixed
MoveJ #joint3
MoveJ #joint4
```

처음 Step 에서 최대 속도의 10%로 설정, #joint1 으로 10% 속도로 이동

#joint2 로 default speed 로 이동, 20mm/sec 속도로 전부 고정 설정

#joint3 로 20mm/sec 로 이동, #joint4 로 20mm/sec 로 이동.

(주의) Speed 를 % 로 사용시 가/감속 시간으로 인해 동작 시간이 속도의 배수단위로 변하지 않습니다. 예를 들어 Speed 100 으로 p1 에서 p2 까지 소요 시간이 2sec 라고 하면 Speed 50 으로 p1 에서 p2 까지 소요시간이 4sec 가 되지 않습니다.

동작 시간을 Speed 의 배수단위로 변하게 하려면 robot config 파일에서

SPEED_RATE_TYPE = 1 로 설정하면 됩니다(default 값이 0 이다). 주의할 것은 이때 로봇의 가/감속 시간이 변하게 됩니다.

SubAbort – Subtask를 정지합니다.

Description:

Subtask 를 정지합니다.

Syntax:

SubAbort subtask

Arguments: SubAbort, subtask

Data type : Value Number

정지하려고 하는 Subtask number(1~4)

Example:

SubAbort 1

Subtask 1 번을 정지합니다.

SubExecute – Subtask를 실행합니다.

Description:

Subtask 를 실행합니다.

Syntax:

SubExecute task number, program name, cycle, step

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

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

실행하려고 하는 Subtask number(1~4)

Sub 프로그램 이름

실행 Cycle 개수

시작 행

Example:

SubExecute 1, sub1, 10, 2

Subtask 1 번에서 sub1 프로그램을 2 번째 행부터 10cycle 실행합니다.

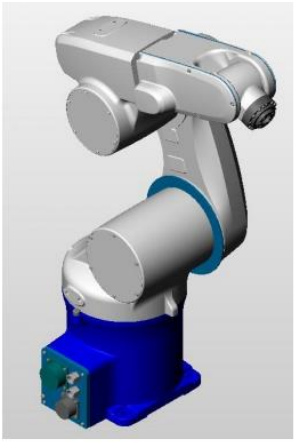
UWRIST/DWRIST – ROBOT의 WRIST의 형상을 결정합니다

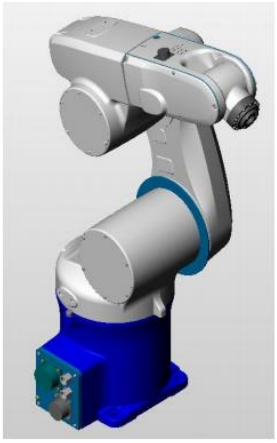
Description:

로봇의 손목 형상이 위로 구부러진 손목이라면 UWRIST, 로봇의 형상이 아래로 구부러진 손목이라면 DWRIST 를 선택합니다, Joint 4, 5 로 결정합니다.

Syntax:

UWRIST/DWRIST





Wait – 조건이 성립하거나 설정한 시간이 끝날 때까지 대기합니다.

Description:

입력된 Digital Input 에 Signal 이 입력 되거나 설정한 시간이 끝날 때까지 대기합니다, Signal 입력을 받으면 대기 상태에서 해제되며 Result Flag 가 True 가 됩니다, 하지만 TimeOut 으로 대기가 해제가 되면 Result Flag 가 False 가 됩니다, Result Flag 를 통해 Signal 과 TimeOut 을 구분할 수 있습니다.

Syntax:

WAIT condition1, time1, result

Arguments: condition1, time1, result

Data type : Digital Input Number or function sig, Value Number, Result Number
DI 는 1001~1128 까지 사용할 수 있습니다, sig 함수를 사용할 수 있습니다,
Value Number 는 sec 단위 입니다, Result 는 0 과 -1 이 출력되며
if 문과 함께 사용합니다

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
```

DI 1001 신호를 3.5 초 간 기다립니다, Signal 이 On 되면 result1 은 true
If 문은 result1 상태를 확인하고 True 일 때 MoveJ #p1 을 실행
DI 1001, 1002 신호를 3.5 초 간 기다립니다,
DI 1001 과 1002 의 Signal 이 전부 On 이 되면 result2 는 true
그 다음 If 문에서 result2 상태를 확인하고 True 일 때 MoveJ #p2
을 실행하고 동작 완료 합니다.

WaitTime – 입력된 시간 동안 대기합니다.

Description:

입력된 시간 동안 다음 명령을 실행하지 않고 대기합니다

Syntax:

WaitTime timevalue

Arguments: timevalue

Data type : Time Number

timevalue 를 설정한 시간 만큼 대기한다.

WaitSig – Signal 조건이 성립할 때까지 대기합니다.

Description:

입력된 Digital Input 에 Signal 이 입력될 때까지 대기합니다,
Signal 입력을 받으면 대기 상태에서 해제 되어 다음 Step 으로 이동합니다.

Syntax:

Wait Sig(DI, DI), waittime, result

Arguments:

Data type : Digital Input Number, Bool Type
DI 는 1001~1128 까지 사용할 수 있습니다.
waittime 는 대기하는 시간을 입력합니다.
result 는 대기 시간동안 DI 신호를 기다리다가 결과 값이 입력됩니다.
signal 이 입력되면 true, 입력되지 않으면 false

Example:

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

DI 1001, 1002 에 Signal 이 입력 될 때까지 다음 Step 으로 이동하지 않고 멈춰 대기합니다, DI 1001, 1002 에 Signal 이 입력 되면 result 의 값이 true 가 되어 if 조건이 성립하여 MoveJ #p1 을 실행합니다.

CL Programming Manual

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